Studying the Benefits of Using UML on Software Maintenance: an Evidence-Based Approach

Ana M. Fernández Sáez

October 2018

Supervisors: Prof Dr. Joost Kok (University of Leiden, The Netherlands) Prof. Dr. Michel R.V. Chaudron (Gothenburg University, Sweden) Prof. Dr. Marcela Genero (University of Castilla-La Mancha, Spain)
Images used in the introduction of each chapter:

Chapter1: https://goo.gl/aP2Sxw
Chapter2: https://goo.gl/YXsPGg
Chapter3: https://goo.gl/gJZpYK
Chapter4: https://goo.gl/KmYrTK
Chapter5: https://goo.gl/p6Sy5H
Chapter6: https://goo.gl/qFNvEU
Chapter7: https://goo.gl/uRq4Vv
# CONTENTS

## CHAPTER 1. INTRODUCTION

1.1. INTRODUCTION ........................................................................................................... 2
1.2. RESEARCH METHODOLOGY ..................................................................................... 9
1.3. CONTEXT OF THE THESIS ....................................................................................... 17
1.4. CONTRIBUTIONS OF THIS THESIS .......................................................................... 20
1.5. OUTLINE OF THE STRUCTURE OF THIS THESIS .................................................. 20

## CHAPTER 2. STATE-OF-THE-ART

ABSTRACT .......................................................................................................................... 26
2.1. INTRODUCTION ........................................................................................................... 26
2.2. RELATED WORK .......................................................................................................... 29
2.3. PLANNING .................................................................................................................. 30
2.4. CONDUCTING THE REVIEW ..................................................................................... 35
2.5. REPORTING RESULTS AND DATA SYNTHESIS ...................................................... 40
2.6. DISCUSSION ............................................................................................................. 59
2.7. THREATS TO VALIDITY ............................................................................................ 62
2.8. CONCLUSIONS .......................................................................................................... 63

## CHAPTER 3. DOES THE LEVEL OF DETAIL (LOD) OF UML DIAGRAMS INFLUENCE SOFTWARE MAINTENANCE?

ABSTRACT .......................................................................................................................... 68
3.1. INTRODUCTION ........................................................................................................... 68
3.2. RELATED WORK .......................................................................................................... 70
3.3. THE FAMILY OF EXPERIMENTS .............................................................................. 76
3.4. RESULTS ..................................................................................................................... 91
3.5. SUMMARY AND DISCUSSION OF THE DATA ANALYSIS .......................................... 105
3.6. IMPLICATIONS OF THE STUDY .............................................................................. 109
3.7. THREATS TO VALIDITY ............................................................................................ 111
3.8. CONCLUSIONS AND FUTURE WORK ..................................................................... 113

## CHAPTER 4. ARE FORWARD DESIGNED OR REVERSE-ENGINEERED UML DIAGRAMS MORE HELPFUL FOR CODE MAINTENANCE?

ABSTRACT .......................................................................................................................... 118
4.1. INTRODUCTION ........................................................................................................... 118
4.2. RELATED WORK .......................................................................................................... 121
4.3. DESCRIPTION OF EXPERIMENT ............................................................................. 128
4.4. RESULTS ..................................................................................................................... 139
4.5. IMPLICATIONS OF THE STUDY .............................................................................. 151
4.6. THREATS TO VALIDITY ............................................................................................ 152
4.7. CONCLUSIONS AND FUTURE WORK ..................................................................... 154
CHAPTER 5. WHAT IS THE PERCEPTION OF INDUSTRY PROFESSIONALS AS REGARDS THE VALUE OF USING UML IN SOFTWARE MAINTENANCE? ................................................................. 157
  ABSTRACT ........................................................................................................... 158
  5.1. INTRODUCTION ............................................................................................. 158
  5.2. SURVEY DESCRIPTION ............................................................................... 159
  5.3. RESULTS ........................................................................................................ 161
  5.4. THREATS TO VALIDITY ............................................................................... 175
  5.5. CONCLUSIONS AND FUTURE WORK ...................................................... 175

CHAPTER 6. WHAT IS THE IMPACT OF USING UML IN SOFTWARE MAINTENANCE IN TERMS OF ITS USE IN A SOFTWARE PROJECT? ................................................................. 179
  ABSTRACT ........................................................................................................... 180
  6.1. INTRODUCTION ............................................................................................. 180
  6.2. RELATED WORK ........................................................................................... 182
  6.3. RESEARCH METHOD ................................................................................... 185
  6.4. RESULTS ........................................................................................................ 196
  6.5. RECOMMENDATIONS .................................................................................. 234
  6.6. SUMMARY OF RESULTS BY RESEARCH QUESTION ............................... 237
  6.7. THREATS TO VALIDITY ............................................................................... 242
  6.8. CONCLUSIONS ............................................................................................. 243
  6.9. FUTURE WORK .............................................................................................. 244

CHAPTER 7. CONCLUSIONS .................................................................................. 247
  7.1. ACHIEVEMENT OF GOALS AND SUMMARY OF FINDINGS ............... 248
  7.2. CONTRIBUTIONS OF THE PHD THESIS ................................................... 254
  7.3. FUTURE RESEARCH LINES ....................................................................... 258
  7.4. PUBLICATIONS ............................................................................................. 259

BIBLIOGRAPHY ..................................................................................................... 263

APPENDIX A. LIST OF PRIMARY STUDIES ......................................................... 275

APPENDIX B. DEFINITIONS OF MEASURES ...................................................... 279

APPENDIX C. THE SEARCH STRINGS ................................................................. 283

APPENDIX D. EXAMPLES OF LOW AND HIGH LOD DIAGRAMS ............... 287

APPENDIX E. INTERVIEW QUESTIONNAIRE ..................................................... 289

APPENDIX F. BACKGROUND INFORMATION RELATED TO EACH INTERVIEWEE .......................................................................................................................... 291

CURRICULUM VITAE ............................................................................................. 295

SUMMARY ............................................................................................................. 297

SAMENVATTING .................................................................................................. 301

ACKNOWLEDGMENTS ......................................................................................... 305
INDEX OF FIGURES

Figure 1.1. An example of a UML class diagram .................................................. 3
Figure 1.2. An example of a UML use case diagram ............................................. 4
Figure 1.3. An example of a UML sequence diagram .......................................... 4
Figure 1.4. Research plan .................................................................................. 7
Figure 1.5. Phases of a Systematic Mapping Study .............................................. 10
Figure 1.6. Global view of the experiment process ............................................. 13
Figure 1.7. Research project timeline during PhD thesis development ............... 20
Figure 1.8. Chronology of publications and stays of this PhD Thesis ................. 21
Figure 2.1. Selection process ........................................................................... 39
Figure 2.2. Factors that influence the maintainability of a system ..................... 56
Figure 2.3. Number of papers per year ................................................................. 57
Figure 3.1. Chronology of the family of experiments ......................................... 77
Figure 3.2. Example of Understandability task .................................................. 85
Figure 3.3. Example of Modifiability tasks for System B .................................... 86
Figure 3.4. Meta-analysis of \( U_{Effic} \) .................................................................. 97
Figure 3.5. Meta-analysis of \( U_{Effec} \) .................................................................. 97
Figure 3.6. Meta-analysis of \( M_{Effec} \) ................................................................. 98
Figure 3.7. Meta-analysis of \( M_{Effic} \) ................................................................. 98
Figure 3.8. Interaction between LoD and System for \( U_{Effec} \) ......................... 99
Figure 3.9. Interaction between LoD and System for \( U_{Effic} \) ......................... 99
Figure 3.10. Interaction between LoD and System for \( M_{Effec} \) ...................... 100
Figure 3.11. Interaction between LoD and System for \( M_{Effic} \) ...................... 100
Figure 3.12. Subjects' perception of the experiment ......................................... 102
Figure 3.13. Subjects’ difficulties when reading diagrams ................................. 103
Figure 3.14. Subjects’ difficulties when reading source code ............................ 103
Figure 3.15. Subjects’ opinion of LoD ............................................................... 104
Figure 4.1. Chronology of the family of experiments ....................................... 129
Figure 4.2. Example of adaptive maintenance task ......................................... 134
Figure 4.3. Example of corrective maintenance task ....................................... 134
Figure 4.4. Answer sheet related to the element “Class”, filled with an example 135
Figure 4.5. Meta-analysis results for Maintainability Effectiveness (\( M_{Effec} \)) ... 143
Figure 4.6. Meta-analysis results for Maintainability Efficiency (\( M_{Effic} \)) ....... 143
Figure 4.7. Interaction between Origin and Ability for Maintainability Effectiveness (MEffec) ......................................................... 144
Figure 4.8. Interaction between Origin and Ability for Maintainability Efficiency (MEffic) ............................................................ 145
Figure 4.9. Subjects' answers as regards adequacy of time provided ........... 146
Figure 4.10. Subjects' answers as regards difficulty of task ..................... 146
Figure 4.11. Subjects' answers as regards adequacy of the LoD .................. 146
Figure 4.12. Subjects' answers as regards difficulties when reading diagrams. 147
Figure 4.13. Subjects' answers as regards usefulness of class diagram used. 147
Figure 4.14. Subjects' answers as regards usefulness of sequence diagrams used 148
Figure 5.1. Use of UML diagrams during software maintenance per role ........ 163
Figure 5.2. Use of UML diagrams during software maintenance by educational level ........................................................................ 163
Figure 5.3. Use of UML diagrams by software development methodology ...... 168
Figure 5.4. Relationship between size of company and use of UML in software maintenance ........................................................................ 170
Figure 5.5. Relationship between role of software in the company and use of UML in software maintenance ....................................................... 172
Figure 5.6. UML diagrams used during software maintenance .................. 173
Figure 5.7. UML diagrams created during software development ............. 174
Figure 6.1. Matrix structure of the ICT organization ................................ 191
Figure 6.2. Flow of documentation in the company and responsibility roles .... 192
Figure 6.3. The type of case study based on Yin’s definition (Yin, 2002) .......... 193
Figure 6.4. Case study execution and analysis procedure .......................... 196
Figure 6.5. "Baseline" theory ................................................................. 198
Figure 6.6. Legend of the theory-diagrams .............................................. 198
Figure 6.7. Responses regarding the main reasons for the use of UML ........ 201
Figure 6.8. Theory summarizing the purposes of the use of UML ............. 203
Figure 6.9. Summary of findings about cost-types ..................................... 205
Figure 6.10. Influences of the SE process in the modelling approach ........ 209
Figure 6.11. Summary of reasons for not maintaining documentation, and risks involved in not maintaining documentation .......................... 214
Figure 6.12. Summary of findings about Practice ..................................... 222
Figure 6.13. Summary of findings about Style ......................................... 224
Figure 6.14. Needs of modelling tools detected in the case study .............. 230
Figure 6.15. Influence of outsource/offshored maintenance on the documentation approach ................................................................. 231
Figure 6.16. Influence of legacy systems on the modelling approach................. 232
Figure 7.1. Contextual factors that affect the adoption of UML diagrams........... 249
Figure 7.2. Main factors studied and relations among them............................. 254
Figure 7.3. Overview of the impacts studied on this PhD Thesis....................... 255
INDEX OF TABLES

Table 1.1. Projects that form the context of the thesis. .............................................. 18
Table 2.1. Search string. ........................................................................................................ 32
Table 2.2. Summary of search strategy. .................................................................................. 32
Table 2.3. Summary of selection strategy. .............................................................................. 33
Table 2.4. Data extraction form. ............................................................................................ 34
Table 2.5. Quality checklist. ................................................................................................... 36
Table 2.6. Systematic Mapping Study outline. ....................................................................... 38
Table 2.7. Results per type of diagram. .................................................................................. 41
Table 2.8. Variables and measures for class diagrams. ......................................................... 42
Table 2.9. Variables and measures for statechart diagrams. ............................................... 44
Table 2.10. Variables and measures for sequence diagrams. .............................................. 44
Table 2.11. Variables and measures for collaboration diagrams. ....................................... 44
Table 2.12. Variables and measures for activity diagrams. .................................................. 45
Table 2.13. Variables and measures for use case diagrams. ................................................. 45
Table 2.14. Variables and measures for the UML diagrams, in general. ............................. 45
Table 2.15. Variables and measures for the system (source code and diagrams)... 46
Table 2.16. Results per duration. .......................................................................................... 47
Table 2.17. Results per empirical method. .......................................................................... 48
Table 2.18. Results per context. ............................................................................................ 49
Table 2.19. Results per type of subject................................................................................ 49
Table 2.20. Results per object to maintain. ........................................................................... 50
Table 2.21. Results per type of system. .................................................................................. 50
Table 2.22. Treatments in the empirical studies. ................................................................... 51
Table 2.23. Quality of primary studies. .................................................................................. 53
Table 2.24. Number of papers per type of publication.......................................................... 58
Table 2.25. Diagrams obtained from RE. .............................................................................. 58
Table 3.1. Summary of related works........................................................................................ 74
Table 3.2. Levels of detail in UML diagrams......................................................................... 78
Table 3.3. Description of the systems received. ................................................................. 79
Table 3.4. Experimental design. ............................................................................................ 84
Table 3.5. Summary of modifiability tasks. .......................................................................... 86
Table 3.6. Post experiment questions. ................................................................................... 87
Table 3.7. Descriptive statistics for $UEffec$. ....................................................................... 91
Table 3.8. Descriptive statistics for $UEffic$. ...................................................................... 92
Table 3.9. Descriptive statistics for MEffec. ......................................................... 92
Table 3.10. Descriptive statistics for MEffic. ............................................................. 92
Table 3.11. Wilcoxon tests results for UEffec. ............................................................ 93
Table 3.12. Wilcoxon tests results for UEffic. ............................................................ 94
Table 3.13. Wilcoxon tests results for MEffec. ........................................................... 95
Table 3.14. Wilcoxon tests results for MEffic. ........................................................... 95
Table 3.15. U-Mann Whitney tests results for the influence of Order. ..................... 100
Table 3.16. U-Mann Whitney tests for the influence of Ability. ............................. 101
Table 3.17. Summary of results of the family of experiments. ............................. 106
Table 4.1. Summary of the related work. ................................................................. 126
Table 4.2. Metrics of the software system used in the family of experiments. .... 130
Table 4.3. Experimental design. ............................................................................. 133
Table 4.4. Summary of maintenance tasks. ............................................................. 135
Table 4.5. Post-Experiment Survey. ..................................................................... 136
Table 4.6. Descriptive statistics for MEffec. ............................................................. 140
Table 4.7. Descriptive statistics for MEffic. ............................................................. 140
Table 4.8. Statistical relation between Origin of diagram (RE/FD) and Maintainability Effectiveness (MEffec). ................................................................. 141
Table 4.9. Statistical relation between Origin of diagram (RE/FD) and Maintainability Efficiency (MEffic). ................................................................. 141
Table 4.10. Statistical relation between Ability and Maintainability Effectiveness (MEffec) and Efficiency (MEffic). ............................................................. 144
Table 4.11. Usage of UML diagrams for solving tasks. ..................................... 149
Table 4.12. Summary of the results of the family of experiments. .................... 150
Table 5.1. Frequency of use of source code only. ................................................. 165
Table 5.2. Effort of consulting UML diagrams for UML users, and effort of consulting documentation for non UML group................................. 167
Table 5.3. Relationship between geo-distribution of the maintenance teams and use of UML in software maintenance......................................................... 169
Table 5.4. Relationship between geo-distribution of companies and use of UML in software maintenance................................................................. 170
Table 5.5. Relationship between size of ICT department and use of UML in software maintenance ................................................................. 171
Table 5.6. Relationship between team size and use of UML in software maintenance ........................................................................................................ 171
Table 5.7. Relationship between size of system maintained and use of UML in software maintenance ................................................................. 172
Table 6.1. Summary of related work. ................................................................. 186
Table 6.2. Cost factors related to the use of UML. ........................................ 204
Table 6.3. Summary of cost of modelling or not modelling............................ 207
Table 6.4. Presence of diagrams in the documentation. ................................. 216
CHAPTER 1. INTRODUCTION
Chapter 1: Introduction

1.1. INTRODUCTION

This PhD thesis presents a collection of published papers. This chapter is therefore devoted to presenting the relevance of the theme of this thesis and the problem statement which motivated us to research it. Our research objectives are also described, along with an explanation of the research methods employed and the context of this dissertation. Finally, the structure of this thesis is outlined and the main contributions are summarised.

1.1.1. Relevance of modeling in software projects

The increasing complexity of software projects (van Vliet, 2008) has led to the emergence of modelling languages with which to increase the understanding between customer and developer (in the analysis phase), improve communication among team members (Nugroho and Chaudron, 2009), and increase the understanding of how software works (both in the development and the maintenance phases). In practice, the quality of software models is therefore primarily important for software architects, developers, and maintainers. Software architects, for example, need to ensure that high level design decisions are appropriately translated into detailed designs. Software developers need comprehensible and consistent models on the basis of which the implementation is constructed. Finally, software maintainers depend on models that have a sufficient correspondence with the actual implementation in order for them to perform maintenance activities efficiently.

Modelling is an activity that is carried out to create representations of the software systems, which involves creating technical solutions that meet the requirements of a system. Good software models perfectly capture user requirements and translate them into technical designs. However, a perfect match to requirements is not the only aspect that is crucial for software models. Models of systems are also characterised by their syntactic and semantic quality. Paying attention to the quality of software models is important because it might determine the quality of the final software product. This is particularly true when software models are used as a foundation on which to develop the actual software system.

In order to increase the quality of the final software product, it is important to pay attention to the software representation, and it is therefore advisable for the software representations to conform to a standard specification of a modelling language. The Unified Modeling Language (UML) originated in 1997 in order to be used as a graphical modelling language geared towards modelling object-oriented software systems (Blaha and Rumbaugh, 2004). It was proposed as a standard for the Object Media Group (OMG) some years later (ISO/IEC, 2005). The current version of UML, version 2.4.1 (OMG, 2011), has 14 diagram types that are divided into two categories, namely structural and behavioral diagrams. UML has been accepted in industry to such an extent that it is now considered as an industrial standard for software modelling (Erickson and Siau, 2007; Grossman et al., 2005). Of the 14 possible diagrams, those most commonly used are the following (Agner et al., 2013; Dobing and Parsons, 2006; Grossman et al., 2005; Hutchinson et al., 2014):
The class diagram (Figure 1.1): A diagram that outlines the relations between the classes (entities) in a system. This diagram is widely used because it describes the structural aspect of a system. It shows how classes in the system are interrelated, and their static relations. It also captures constraints regarding the way in which classes interact with each other. Depending on their level of detail, this type of diagram may additionally show attributes and operations of classes.

Figure 1.1. An example of a UML class diagram.

The use case diagram (Figure 1.2): A diagram that depicts an overview of a system in terms of the relation between various usages of a system. This diagram is widely used because it captures the functionality that needs to be delivered by a system. This functionality is visualised in terms of horizontal ellipses that are connected to actors that perform or execute the functionality.

The sequence diagram (Figure 1.3): A behavior diagram in which a sequence of messages between instantiations of objects is modelled. This diagram is widely used because it captures interactions of objects in order to deliver a particular functionality.

These three diagrams provide three different points of view of the system modelled. These views complement each other, offering different levels of detail or providing different purposes for each view.
In practice, UML is used in a variety of ways. Ordered from informal to formal in the sense of diagram completeness and adherence to the UML standard, these are:

- **As a sketch:** Developers can use the notational elements of UML to quickly draw part of a system for comprehension and communication purposes. For sketches, UML elements such as classes and actors might be used in combination with informal or domain-specific constructs (Cherubini et al., 2007).

- **For the communication of system design:** Modelling parts of a system makes it possible to explain, for example, how a system component is supposed to function. Depending on how much of a system is modelled, this approach can be a form of model-centric development.

- **As a blueprint:** In this case, most of the system analysis and design has already taken place and the resulting set of UML diagrams is then used for implementation. This type of development approach is referred to as model-centric development. UML is likely to be used as a blueprint in the context of Global Software Development during which design and coding activities take place at different
geographical locations and there is a need for agreement on the details of the system.

- As a programming language: UML diagrams can be used as a basis for code generation. The UML diagrams must strictly adhere to a predefined syntax. This type of development approach is referred to as Model-Driven Development.

1.1.2. Problem statement

Including modelling as part of software development appears to have various benefits. Why then is it that not all companies use software modelling? One of the main reasons is that it requires up-front investments. From an economic point of view, any type of investment must be justified in terms of how much payback there will be at a later stage. This being the case, in the context of software projects, investment in modelling should be justified by benefits, such as improved productivity and improved product quality, which can be seen later during software development or maintenance. When such benefits are not tangible or foreseeable, modelling becomes a practice without clear added value for the system being developed.

The benefits of using software documentation to comprehend and modify source code have been widely studied (Abbes et al., 2011; de Souza et al., 2005; Tilley and Huang, 2003). The authors of (Tryggeseth, 1997) report that having documentation available during system maintenance reduces the time needed to understand how to perform maintenance tasks by approximately 20 percent. The availability of good documentation is also believed to minimise the loss of information and misinterpretation as regards communicating decisions made during the complete software life cycle. Software documentation might also increase productivity (Arisholm and Briand, 2006).

The big issue regarding software modelling in fact resides exactly in the assumption that it will provide software projects with quantifiable benefits. The problem, therefore, is how we can investigate and prove whether or not modelling, or some specific characteristics of modelling, provide any benefits during software development and maintenance. As long as this question remains unanswered, it will be difficult to motivate and justify modelling activities in real software projects. This thesis therefore contributes to partially answering these open questions by focusing the research on the benefits of using UML modelling during software maintenance. We have focused on the UML as a modelling language because, as mentioned previously, UML has become the de facto standard graphical modelling notation used in software documentation. We have also focused on the maintenance phase because it is the phase of the software life cycle which consumes the majority of software development resources. As explained in (Pressman, 2005) and (Glass, 2002): “Maintenance typically consumes 40 percent to 80 percent of software costs. Therefore, it is probably the most important life cycle phase of software” and “60 percent of the budget is spent on software maintenance, and 60 percent of this maintenance is to enhance”. Enhancing old software is, therefore, a huge undertaking.
1.1.3. Objective of the Thesis

The main goal of our research is to investigate the benefits of modelling in software maintenance. We particularly focus our attention on UML as a modelling language since, as explained previously, it is widely used in industry.

Our main research question is, therefore:

“What is the impact of using UML in software maintenance?”

This main research question was used as the basis to begin reviewing the existing literature (Figure 1.4) in order to discover and analyse what has already been done in this field and to identify the remaining gaps which are potential areas for further investigation.

As the main research question is too general, and after considering the main findings obtained through an exhaustive literature review, three research questions were initially formulated (Figure 1.4):

- **RQ1.** What is the perception of professionals in industry as regards the value of using UML in software maintenance?
- **RQ2.** Does the level of detail (LoD) of UML diagrams influence software maintenance?
- **RQ3.** What is the impact of using UML in software maintenance in terms of its use in a software project?

Moreover, new insights obtained during this long-term study led us to define a new research question:

- **RQ4.** Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?

These research questions were formulated in order to observe the impact of UML modeling on software maintenance from different perspectives and in different contexts, such as the academic context, by collecting evidence in a controlled environment from the point of view of students who will be the future professionals, and in an industrial context by collecting data from the point of view of current software engineers.

Bearing in mind that they cover a wide spectrum of research, the questions formulated were addressed by following a mixed-method approach (Creswell, 2013). We were of the opinion that considering different perspectives and combining multiple research methods in order to answer the principal question would enable us to obtain a more comprehensive understanding of the impact of UML modeling on software maintenance.
Chapter 1: Introduction

Study of the state of the art: Systematic Mapping Study

- Which diagrams are most frequently used in studies concerning UML and maintenance?
- Which variables are investigated in the empirical studies (experiments, case studies and surveys)?
- What is the state-of-the-art in empirical studies of UML maintenance?
  - How is the influence of UML on maintenance studied?
  - Where are the empirical studies carried out?
  - Who is evaluated in the empirical studies?
  - What is maintained in the empirical studies?

RQ1: What is the perception of professionals in industry as regards the value of using UML in the maintenance?
RQ2: Does the Level of Detail (LoD) of UML diagrams influence the maintenance of the code?
RQ3: What is the impact of using UML in software maintenance in terms of its use in a software project?
RQ4: Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?

Survey
Family of experiments
Case study
Family of experiments

Figure 1.4. Research plan.
1.1.4. Relations between research questions and the main goal

It is important to explain how each research question contributes to obtaining the answer to the main research question:

**RQ1. What is the perception of professionals in industry as regards the value of using UML in software maintenance?** This provides a subjective idea of the success of using UML in software maintenance. It also provides insights into how maintainers believe that using UML affects their daily work.

**RQ2. Does the level of detail (LoD) of UML diagrams influence software maintenance?** We first considered performing an experiment to discover whether or not maintenance is influenced by the use of UML, but the results of the preliminary study showed us that this topic had already been studied in (Dzidek et al., 2008). While various benefits of using modeling in software development and maintenance are commonly reported, software organizations seem to fear the effort involved in the creation of models are part of documentation. An investment needs a payback. In order not to use more resources for the creation of the UML models than the paybacks that might originate from their use, it would be necessary to define the proper Level of Detail (LoD) of UML diagrams as part of the documentation. To the best of our knowledge, the influence of LoD has already been studied, but in relation to the development (Nugroho, 2009; Nugroho et al., 2008) rather than the maintenance stage. It would be also desirable to help organizations to define the proper LoD necessary to update the UML documentation in order for it to be synchronized with the source code and thus attain understandability benefits during further maintenance. This study, therefore, provides some insights into which LoD in a UML diagram it is best to use in a specific context (novice developers or academic context).

**RQ3. What is the impact of using UML in software maintenance in terms of its use in a software project?** This provides an industrial point of view concerning the influence of UML on maintenance tasks. We began performing some interviews related to RQ3, and we found that maintenance tasks are sometimes supported by UML diagrams built at the beginning of the development and that in other cases the diagrams are obtained by means of reverse engineering processes. In order to start our study in an academic context, and whilst waiting to attain an industrial case study that would allow us to work on RQ3, we decided to tackle RQ4.

**RQ4. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?** As explained previously, this study was carried out in base to results and insights of RQ2 and RQ3, in order to investigate whether the origin of the UML diagrams influences maintenance tasks. Reversed Engineered (RE) diagrams are diagrams with a very high LoD since they represent all the elements in the source code. The Forward Designed (FD) diagrams might also be considered as diagrams with a high LoD depending on their content (class names, attributes, operations and relationships for class diagram; and lifelines, messages and parameters of messages in sequence diagrams). However, FD diagrams more commonly do not represent all the elements in the source code. The key characteristics of the comparison of RE and FD diagrams is: 1) RE diagrams are complete and close to source code; and 2) man-made
FD diagrams are almost never complete, but their strength seems to be the selective inclusion of information about the system and modest complexity of the diagrams. This study presents the results of a family of experiments carried out in an academic context in order to investigate how maintainers work with diagrams generated by means of a Forward-Design process in comparison to diagrams generated using Reverse Engineering techniques.

1.2. RESEARCH METHODOLOGY

As our approach for addressing the research questions in this dissertation is empirical in nature, in this section we shall introduce the main characteristics of the empirical research methods used to obtain empirical evidence that will allow us to answer the proposed research questions (Figure 1.4). A more detailed explanation of them will be provided in the chapter in which each of them is used.

We have selected the most appropriate research method for each research question, signifying that various research methods have therefore been used. This is known as the mixed-method approach (Creswell, 2013). Mixed methods research is an approach that combines quantitative and qualitative research methods in the same research inquiry. Such work can help develop rich insights into various phenomena of interest that cannot be fully understood using only a quantitative or a qualitative method. A method that is applied in isolation has its own strengths but also its own weakness, so a combination of research methods, and particularly a triangulation of qualitative and quantitative data, helps researchers develop a more profound understanding of a phenomenon.

We began collecting the existing empirical evidence on the topic addressed by means of a systematic mapping study; in this case, we followed a combination of the guidelines provided by Petersen et al. (Petersen et al., 2008) and Kitchenham et al. (Kitchenham et al., 2015). We later carried out several empirical studies by considering the following research methods: survey (Fink, 2002; Pfleeger and Kitchenham, 2001), interview (Steinar, 2007) (as a means to perform surveys), families of experiments (Wohlin et al., 2012) and a case study (Runeson et al., 2012).

1.2.1. Systematic mapping study (SMS)

In order to identify the existing research in relation to the topic being studied in this thesis, we first performed an SMS. Systematic Mapping Studies (SMSs) are studies, whose principal objective is to provide a global view of the subject of interest (with or without an empirical approach) and to identify the quantity and type of research available and the results obtained from it. This makes it possible to identify those subjects on which there is little empirical evidence and for which it is necessary to carry out more empirical studies. SMSs usually serve as a basis upon which researchers can carry out future investigations, always supposing that they have been carried out in a rigorous manner (Kitchenham, B. et al., 2011). The well-defined methodology makes it less likely that the results obtained the literature selected will be biased, although it does not protect against publication bias in the primary studies. In
the case of quantitative studies, it is possible to combine data using meta-analytic techniques. This increases the likelihood of detecting real effects that individual smaller studies are unable to detect. The major disadvantage of SMSs is that they require considerably more effort than traditional literature reviews.

SMSs are widely accepted in software engineering (SE), although this form of study appears to be less widely used in other disciplines. This may reflect the nature of SE, which is not a truly “empirical” discipline, although it is slowly moving in that direction. We may, therefore, often have both very limited knowledge about how widely a topic has been studied, and also relatively few studies that are empirical in form. With regard to this, empirical studies may well be reported in many different venues, meaning that we need to carry out a wide search to find all relevant material. As stated in (Kitchenham et al., 2015), an SMS is a useful method for a PhD literature review.

An SMS comprises three principal activities, each of which consists of various sequential tasks (Figure 1.5), of which it is usually necessary to carry out several iterations. A brief description of each of them is provided below:

1. **Planning the review**: Planning is crucial, since the correct development of the SMS will depend on the decisions made during this activity. The principal objective of planning is to specify all those aspects that will make the review systematic and rigorous, avoiding as far as possible all bias and ambiguity and detailing what is denominated as the review protocol.
2. **Conducting the review**: All that was previously planned in the protocol is put into practice in this activity, and the final results that will answer the research questions are obtained. It is also fundamental to document all the incidents that occur and the decisions made during the tasks carried out when executing the review. This will make it possible to replicate the SLR and will signify that all the decisions made will be available for external reviewers and anyone else who might wish to use the results obtained.

3. **Reporting the review**: Finally, a report is written that reflects the entire review process, considering the means used to share the information that was selected when defining the protocol. The recommended structure and content when writing the report of the review are presented in (Kitchenham and Charters, 2007). Space restrictions may make it necessary to complement the publication with a technical report whose website is shown in the report. This makes it possible to include relevant but highly extensive information, such as the complete definition of the protocol, the list of primary studies, the data extraction forms filled out with information on the case study, the evaluation of the quality of each study, etc.

### 1.2.2. Experiment

An experiment is a formal, rigorous and controlled investigation in which the relation between an effect and a cause is addressed (Wohlin et al., 2012). In software engineering, an experiment is empirical research that handles a variable (denominated as the independent variable or factor) of the environment or phenomenon being studied and measures the effect that it has on another variable denominated as the dependent variable. In a controlled experiment, the effect of a treatment on an experimental group is tested and compared to that of a control group in which the treatment was absent. If the experimental design is in line with the study objective and executed in line with methodological guidelines (as is described in (Wohlin et al., 2012)), the results obtained are regarded as strong evidence. The principal strength of experiments is that they can be used to investigate in which situations statements are correct and may serve to recommend those contexts in which certain standards, methods or tools are useful. One disadvantage of using experiments is that, because of the need to exclude other variables (alternative explanations), the experimental setup has little resemblance to industrial reality. This potentially limits the generalisability of results.

Given that experiments or isolated studies rarely provide sufficient information with which to respond to the questions defined during research, it is of interest to ensure that these experiments form part of a family of studies rather than considering isolated experiments (Basili et al., 1999). These families of experiments may make it possible to extract relevant conclusions concerning the hypotheses that cannot be obtained with individual studies. The studies of which families of experiments are formed should be planned appropriately as replicas of the original experiments.

The synthesis or quantitative addition of the results obtained using families of experiments are continuously carried out by means of meta-analysis techniques (Glass
et al., 1981). Upon combining the results of various experimental studies, the meta-
analysis makes it possible to generate more general and reliable knowledge than that of results obtained by means of individual studies, since that knowledge is supported by a greater quantity of empirical evidence.

If all the studies included in the meta-analysis process are equally precise and employ exactly the same independent variables, it is sufficient to average out the results of each one in order to obtain a final conclusion (Borenstein et al., 2007). However, in practice not all studies have the same precision, and when they are combined it is therefore necessary to assign a greater weight to those that allow more reliable information to be obtained. This is achieved by combining the results using a weighted mean (Cochrane Collaboration, 2003). Furthermore, in order to resolve problems related to the non-uniformity of dependent variables, the results of meta-analysis methods are expressed by means of an index denominated as ‘effect size’, which is an estimator of the size of the relationship between a treatment and a dependent variable (Cochrane Collaboration, 2003) and can be applied to any measurement of the difference between the results in two groups. The objective of a meta-analysis is, therefore, to discover the size of the global effect obtained from the sizes of the effects of each individual study and that will reflect the degree to which the phenomenon being studied is present in the population as a whole.

In order to carry out an empirical study, it is necessary to follow a process which details the activities to be carried out, what must be done and what the inputs and outputs of each activity are. The process (Wohlin et al., 2012) shown in Figure 1.6 is focused on experiments (which is the type of empirical study most frequently used in this thesis), but the same basic activities must be carried out in any type of empirical study. The main difference is how each of the tasks in each activity is carried out, i.e. the design of an experiment. A survey or a case study is different, but all of them must be designed. This means that the basic activities are the same, but each activity must be adapted according to the specific objective of each type of study. The experimental process consists of five activities, each with its corresponding tasks, but this process is not carried out sequentially, as it is not necessary to complete one activity before starting another. In fact, the order of the activities in the process only indicates the order in which each activity should be started. The process is iterative, since in certain cases it is necessary to go back in order to refine a previous activity before continuing with the experiment. It is obviously not possible to go back and refine the objective or the planning once the execution of the experiment has begun.

Each of the activities in the experimental process is briefly described below:

1. **Scope definition**: The objectives of the experiment must be defined in this stage. In order to correctly define all the important aspects of the experiment before going on to its planning and later execution, it is advisable to use the GQM (Goal-Question-Metric) template to define objectives (Basili and Rombach, 1988).
Figure 1.6. Global view of the experiment process.
2. **Planning**: After defining the objectives of the experiment, it is planned in order to attain a clear idea of how it will be carried out.

3. **Operation**: Once the experiment has been planned and designed, it must be carried out in order to collect the data that will later be analysed. This is denominated as the operation of the experiment.

4. **Analysis and interpretation**: After collecting the data obtained once the experiment has been carried out, it is necessary to analyse and interpret them correctly. Three principal aspects should be borne in mind when choosing from among the various analysis techniques: the nature of the data collected, the motive of the experiments and the type of experimental design.

5. **Presentation and diffusion**: When an experiment is carried out, its findings are usually presented, which can be done by communicating it at a congress or in a journal, by means of a report in order to make decisions or as a package that can be used to replicate the experiment, or as educational material. Whatever the case may be, the most important thing is not to forget any important aspects of each of the activities in the experimental process described above. A good documentation of the experiment will allow other researchers to replicate it or use the empirical knowledge contained in it as a basis for other research.

### 1.2.3. Survey

A survey is a comprehensive system that is used to collect information in order to describe, compare or explain knowledge, attitudes and behavior (Pfleeger and Kitchenham, 2001). Survey methods are a well-established social science technique with which to obtain a broad characterisation of a particular issue (Hutchinson et al., 2014) because, as a data collection method, surveys have several crucial potential advantages over less systematic approaches. When they are properly designed, executed, and described, they (1) economically present the characteristics of a large group of objects or respondents and (2) permit an assessment of the extent to which the objects measured or respondents are likely to adequately represent a relevant group of objects, individuals, or social organisms (Diamond, 2011).

In the majority of cases, the data relative to the survey are obtained by means of questionnaires. But questionnaires do not, in themselves, constitute a survey. In fact, a survey is a more complex process that is formed of a series of well defined activities (Kitchenham and Pfleeger, 2008) which are enumerated as follows and which served as guidelines for the development of our survey:

1. **Establish the objectives of the survey**: establishing the objectives is always the first step when conducting a survey or any other type of research. In the specific case of surveys, each objective must be established as a sentence that is relative to the results expected after carrying out the survey itself.

2. **Design the survey**: the two types of survey design most frequently used are: Transversal surveys, which request information from the participants at a particular moment, and 2) Longitudinal surveys, in which the goal is established in the longer
term and the aim is to discover the evolution of a particular population (which may be the same or may change) over time.

3. **Develop the questionnaire:** the objective of a survey is to obtain responses to a series of questions for a particular reason. When designing a survey it is therefore appropriate to base it on the research objective. However, directly translating objectives into questions rarely leads to a useful questionnaire, and if the objective is to attain a genuinely effective survey it is therefore necessary to design the questionnaires used to collect the data correctly and cautiously. Evaluating and validating the questionnaire: once the questions in the questionnaire have been defined, it is necessary to ensure that they will be understood correctly. This can be done by evaluating the probable index of response, and evaluating the reliability and validity of the questionnaire by means of discussion groups and/or pilot studies, and verifying that the data analysis method used will be compatible with the response that will be obtained.

4. **Obtain the data from the survey:** when obtaining the data, it is normally impossible to attain the responses from the entire population implied in the study, and it is therefore necessary to resort to a sample of it in the hope that the responses will represent those that would have been obtained from the complete set.

5. **Analyse the data obtained:** once the survey has been designed and carried out, it is time to analyse the data obtained from it. It is at this moment that the researchers deal with the most important points regarding data analysis. More specifically, they study the validity of the data obtained, the division of the response into homogeneous groups and, finally, the analysis of both the ordinal and the nominal data obtained.

6. **Report the results:** this step is concerned with publishing the information generated when carrying out the survey and the analysis of its results in a journal or at a congress, or even in the form of a technical report associated with a research project.

### 1.2.4. Case study

A case study is “an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified” (Runeson et al., 2012). Case studies are conducted by collecting qualitative information, which is rich but lacks a standardised structure and analysis methods. The nature of a case study is inimitable, and such studies are thus difficult to repeat with other cases. This nature signifies that case studies are sometimes criticised for being impossible to repeat and generalise from, for being biased by researchers, and so on (Runeson and Höst, 2009). For example, case studies do not generate the same results when they are repeated, unlike analytical and controlled empirical experiments. However, the main advantage of case studies is that they provide a more profound understanding of the phenomena under study. In order to mitigate this problem, we followed the guidelines proposed by (Runeson et al., 2012) in order to design, analyse and report the findings.
of the case study presented in this thesis. The most commonly used process is based on the following five activities (Runeson et al., 2012):

1. **Design and plan the case study**: when planning the case study it is necessary to take into consideration at least the following elements: the objective of the study, the case and the unit of analysis, the theory, the research questions, the data collection method and the data selection strategy.

2. **Prepare the data collection**: first, second or third degree data collection techniques are established according to the degree to which the researcher is implied in data collection. Of these, there are interviews, observations, collection of data in files or the calculation of metrics.

3. **Collect the data**.

4. **Analyse and interpret the data collected**: data analysis is carried out differently for qualitative and quantitative data. In the case of qualitative data, an analysis of the descriptive statistics, a correlation analysis, predictive models and a contrast of hypotheses are usually carried out, while in the case of quantitative data, generation techniques and the confirmation of hypotheses are carried out.

5. **Report the results obtained**.

These activities are generally similar to those of other empirical studies, such as the experimental process presented above (see Figure 1.6), although case studies have certain peculiarities.

In order to analyse the data collected as part of the case study presented in this thesis, we used grounded theory (Strauss and Corbin, 1990). Grounded theory (Service, 2009) is an analysis method that is geared towards theory development. Grounded theory is a primary research approach that describes the qualitative sample methods, data collection and analysis, the use of the constant comparison method, the use of theoretical sampling, and the generation of a new theory. It treats the study reports as data that can be analysed in order to generate superior topics and interpretations. The steps that must be followed in order to understand the complete process of grounded theory are (Strauss and Corbin, 1990):

1. The data are collected.
2. The evidence in the data is analysed.
3. If replicas are found in the data, it is possible to start discussing facts.
4. A theory begins to be generated.
5. Concepts that interpret the data are created from the theory.

Interviews were used as a qualitative data collection method as part of the case study. This technique “seeks to cover both a factual and a meaning level” (Kvale, 1996). Interviews are used when data needs to be collected about phenomena that cannot be obtained using quantitative measures. The type of interview used for data collection in the context of the case study is the “qualitative interview” which is “a sort of guided conversation” (McNamara, 1999). The interviews are standardised in the sense that each interviewee is asked similar questions (depending on that person’s role) and open-ended in the sense that there is ample room for interviewees to elaborate. This type of interview is also referred to as semi-structured or focused.
1.3. CONTEXT OF THE THESIS

The author of this thesis has developed it as part of a co-supervision agreement between the University of Castilla-La Mancha (Spain) and the University of Leiden (The Netherlands), receiving financial support from both of them. She is a member of the Alarcos Research Group (http://alarcos.esi.uclm.es), which is led by Professor Mario Piattini. The author of this thesis joined the group as a research fellow in 2007.

She has also developed part of this thesis in collaboration with the FaST-SE research group at the University of Leiden (The Netherlands) which is led by Michel R.V. Chaudron. During 2012, the author was a research member of the FaST-SE.

During the thesis development, she also participated in a collaboration with the Software Engineering Research Laboratory (SERLAB), led by Giuseppe Visaggio, in the Department of Informatics at the University of Bari, Italy.

Finally, the last collaboration was with the Department of Computer Science and Engineering (CSE) of the Chalmers University of Technology at the University of Gothenburg, Sweden.

The afore mentioned research collaborations were part of the following pre-doctoral stays:

- University of Leiden: There were three stays in this university: The first stage was from 11/06/2010 to 31/01/2011, the second was from 01/01/2012 to 22/05/2012, and finally, the third stage was from 30/05/2012 to 29/05/2012.
- University of Bari: This stay lasted from 23/05/2012 to 31/12/2012.
- Chalmers University: There were 2 stays at this university: 1) From 4/12/2013 to 13/12/2013 and 2) From 19/07/2013 to 02/08/2013.

During the rest of the time since 01/06/2010, and discounting the periods of stays abroad, the PhD student has developed her research in the Department of Technologies and Information Systems at the University of Castilla-La Mancha.

Leiden University provided the author of this thesis with economical support by employing her as a Research Assistant during the 20 months mentioned above. She was also employed as a Research Assistant at the University of Castilla-La Mancha during the rest of the time needed to complete this thesis. Some of the research included in this thesis has, therefore, been developed in the context of the following research projects (Table 1.1 and Figure 1.7):
<table>
<thead>
<tr>
<th>Project</th>
<th>Main Researcher</th>
<th>Duration</th>
<th>Participant entities</th>
<th>Founding body</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEDUSAS: Mejora y Evaluación del Diseño, Usabilidad, Seguridad y Mantenibilidad del Software</td>
<td>Mario Piattini Velthuis</td>
<td>01/01/2009 - 01/01/2013</td>
<td>University of Castilla-La Mancha, Alarcos Quality Center, S.L.; Sicaman Nuevas Tecnologías, Audisec S.L., Génesis XXI</td>
<td>CDTI-MICINN ((IDI-20090557)</td>
<td>364.700 €</td>
</tr>
<tr>
<td>MECCA: Metodología para la Evaluación Continua de la Calidad de Artefactos software (Methodology for Continuous Quality Assessment of Software Artifacts)</td>
<td>Marcela Genero Bocco</td>
<td>01/04/2009 - 31/03/2012</td>
<td>University of Castilla-La Mancha</td>
<td>Junta de Comunidades de Castilla- La Mancha (PII2109-0075-8394)</td>
<td>200.000 €</td>
</tr>
<tr>
<td>MAGO /PEGASO: Mejora Avanzada de Procesos Software Globales (Advanced Improvement of Global Software Process)</td>
<td>Mario Piattini Velthuis</td>
<td>01/10/2009 - 30/12/2012</td>
<td>University of Castilla-La Mancha, University of Murcia in coordinated project (PEGASO: Procesos para la mejora del desarrollo GlobAl del Software (Processes to improve global software development))</td>
<td>CDTI-MICINN (TIN2009-13718-C02-01)</td>
<td>489.100 €</td>
</tr>
<tr>
<td>EECCOO: Entornos para la Evaluación Continua de la Calidad de artefactos (Environments for Continuous Evaluation of Artifacts Quality)</td>
<td>Marcela Genero Bocco</td>
<td>01/03/2010 - 28/02/2012</td>
<td>University of Castilla-La Mancha, Alarcos Quality Center</td>
<td>MICINN (TRA2009-0074)</td>
<td>102.245 €</td>
</tr>
</tbody>
</table>

Table 1.1. Projects that form the context of the thesis (part 1/2).
<table>
<thead>
<tr>
<th>Project</th>
<th>Main Researcher</th>
<th>Duration</th>
<th>Participant entities</th>
<th>Founding body</th>
<th>Budget</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORIGIN: Organizaciones Inteligentes Globales Innovadoras (Global Innovative Intelligent Organizations)</td>
<td>Mario Piattini Velthuis</td>
<td>06/04/2010 -31/12/2012</td>
<td>University of Castilla-La Mancha, Sicaman Nuevas Tecnologías, EIDOS, INDRA, SIGTEL</td>
<td>CDTI-MICINN and FEDER IDI-2010043(1-5)</td>
<td>7.312.357 €</td>
</tr>
<tr>
<td>GEODAS: Gestión para el Desarrollo Global de Software mediante Ingeniería de Negocio y Entornos Avanzados de Colaboración (Management for Software Global Development by Business Engineering and Advanced Collaborative Environments)</td>
<td>Mario Piattini Velthuis</td>
<td>01/01/2012 -31/12/2015</td>
<td>University of Castilla-La Mancha, Sicaman Nuevas Tecnologías, ALHAMBRA- EIDOS, INDRA, SIGTEL</td>
<td>MINECO/FEDER (TIN2012-37493-CO3-01)</td>
<td>308.800 €</td>
</tr>
<tr>
<td>IMPACTUM: Aplicación de ISBW en el estudio del IMPACTo de UML en el desarrollo de software en un contexto DSDM-LPS (ISBW application in the study of the impact of UML in software development in a DSDM-LPS context)</td>
<td>José Antonio Cruz-Lemus</td>
<td>29/09/2014-30/09/2015</td>
<td>University of Castilla-La Mancha</td>
<td>Junta de Comunidades de Castilla- La Mancha (PEII11-0330-4414)</td>
<td>50.000 €</td>
</tr>
<tr>
<td>SEQUOIA: Security and QUality in processes with bIg data and Analytics</td>
<td>Marcela Genero Bocco and Eduardo Fernández-Medina Patón</td>
<td>01/01/2016 – 31/12/2018</td>
<td>University of Castilla-La Mancha, University of Alicante, and University of Sevilla</td>
<td>MINECO/FEDER (TIN2015-63502-C3-1-R)</td>
<td>110.100 €</td>
</tr>
</tbody>
</table>

Table 1.1. Projects that form the context of the thesis (part 2/2).
1.4. CONTRIBUTIONS OF THIS THESIS

In essence, the contribution of this study is two-fold. First, this study provides sound empirical evidence about the potential benefits of UML modeling in software maintenance. The findings of this study should also contribute to the body of knowledge, particularly in the field of software engineering. Additionally, from a research perspective, we consider this study as a milestone towards a more comprehensive understanding about the role of modeling in software maintenance.

The second contribution is related to modeling practices. This study provides recommendations concerning best practices of modeling using UML. These recommendations are based on empirical data from an industrial case study as well as expert opinions. Therefore, this study essentially serves as a guide and justification for software engineers to continue using (and updating) the UML models of software systems, and to get the most benefits out of it.

1.5. OUTLINE OF THE STRUCTURE OF THIS THESIS

In this thesis, we have decided to present the results which were obtained from the empirical studies starting with those which took place in academic contexts followed by results obtained from industry. We decided to follow this approach rather than presenting the results in the order of their date of execution, or in the order of the
number of the research question so as to move from less generalizable results to more real ones.

This dissertation is presented as a compendium of papers. Most of the chapters are based on one or more published research papers in order to cover a research question, but only the most complete paper is presented as a chapter in this dissertation.

A summary of the chronology of these published papers and the stays that took place during this research is presented in Figure 1.8:

![Figure 1.8. Chronology of publications and stays of this PhD Thesis.](image)

Five of the published/under submission research papers have been selected to be part of the core chapters of this PhD thesis owing to both their scientific contribution and their relevance. They are either papers that are indexed in the Journal of Citation Report or that appeared in congresses or workshops relevant to the sphere of software modelling.

Bearing the selected papers in mind, the content of what remains of this document is organised as follows:

**Chapter 2: State of the Art.** This chapter contains the Systematic Mapping Study of the relevant literature, i.e. it provides an exhaustive review of the existing literature, which is the background to the thesis. This chapter is based on the following research paper:


**Chapter 3: RQ2. Does the level of detail (LoD) of UML diagrams influence software maintenance?** This chapter summarises the result of a family of experiments performed with the aim of investigating what different levels of detail in a UML representation would influence the tasks performed by software maintainers. In addition to using objective measures applied to the understandability or modifiability of the system being maintained (such as efficiency or effectiveness), subjective perceptions were also obtained.

Two research papers were written on the basis of the research developed to answer RQ2. The first contains the initial experiment of the family of experiments and was published as a conference paper. The second contains the complete family of
experiments and was published as a journal paper. The second, which is the most complete and recent paper, was therefore selected to be presented in this document:


Chapter 4: RQ4. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance? This chapter summarises the results of another family of experiments whose objective was to investigate how maintainers perform their maintenance tasks when using UML diagrams originating from a forward design approach (i.e. they are human-based diagrams) in comparison with the usage of UML diagrams generated by Reverse Engineering tools (i.e. they are automatic diagrams). Objective and subjective measures were again used in order to obtain a broader result.

Two research papers were written on the basis of the research developed to answer RQ4. The first contains the initial experiment of the family of experiments and was published as a conference paper. The second contains the complete family of experiments and was published as a journal paper. The second, which is the most complete and recent, was therefore selected to be presented in this document:


Chapter 5: RQ1. What is the perception of professionals in industry as regards to the value of using UML in software maintenance? This chapter presents the results of a survey with industrial professionals involved in software maintenance projects. They provided their opinion on how documentation (containing or not containing UML diagrams) enables them to perform their maintenance tasks, among other issues. The results of this survey also reflect what kinds of companies are fruitfully using UML diagrams as part of their software documentation.
One paper appeared as a result of the research related to RQ1, which is included in this thesis:


Chapter 6: RQ3. What is the impact of using UML in software maintenance in terms of its effectiveness when used in a software project? This chapter provides an industrial point of view as regards how using or not using UML diagrams as part of the software maintenance documentation might provide a return on the investment made during the design phase of a software life cycle. This study was performed as a case study in an industrial context: the software department of a multinational company. This case study consists of a quantitative analysis of the data extracted from the company repository and a qualitative analysis based on a series of interviews.

Two research papers were published on the basis of this case study, although only the latter is included in this thesis. The first contains the initial results of the case study and it was published as a conference paper. The second contains the complete results of the case study and was sent to a journal in order to be published. The second paper, which is more complete and recent was selected to be presented in this document:


Chapter 7: Conclusions. The last chapter of this thesis summarises the results obtained thanks to the different empirical studies carried out in the previous chapters, with the goal of answering the six research questions formulated. Moreover, the main contributions are highlighted and future work is also outlined.

The following papers were also published in the context of this dissertation, and are related to the current chapter, the introduction. The motivation to undertake the studies shown in this thesis and the research plan are presented in them:


ABSTRACT

Context: The Unified Modelling Language (UML) has, after ten years, become established as the de facto standard for the modelling of object-oriented software systems. It is therefore relevant to investigate whether its use is important as regards the costs involved in its implantation in industry being worthwhile.

Method: We have carried out a systematic mapping study to collect the empirical studies published in order to discover “What is the current existing empirical evidence with regard to the use of UML diagrams in source code maintenance and the maintenance of the UML diagrams themselves?

Results: We found 38 papers, which contained 63 experiments and 3 case studies. Conclusion: Although there is common belief that the use of UML is beneficial for source code maintenance, since the quality of the modifications is greater when UML diagrams are available, only 3 papers concerning this issue have been published. Most research (60 empirical studies) concerns the maintainability and comprehensibility of the UML diagrams themselves which form part of the system’s documentation, since it is assumed that they may influence source code maintainability, although this has not been empirically validated. Moreover, the generalizability of the majority of the experiments is questionable given the material, tasks and subjects used. There is thus a need for more experiments and case studies to be performed in industrial contexts, i.e., with real systems and using maintenance tasks conducted by practitioners under real conditions that truly show the utility of UML diagrams in maintaining code, and that the fact that a diagram is more comprehensible or modifiable influences the maintainability of the code itself. This utility should also be studied from the viewpoint of cost and productivity, and the consistent and simultaneous maintenance of diagrams and code must also be considered in future empirical studies.

Keywords: UML, empirical studies, software maintenance, Systematic Mapping Study, Systematic Literature Review

2.1. INTRODUCTION

UML was first introduced in 1997, and became a de facto standard for the modelling of object-oriented software systems in 2000 (OMG, 1997). It subsequently evolved and the latest version appeared in 2009 (UML 2.3) (OMG, 2010). Owing to the increasing complexity of software projects at the present time, the UML has emerged as a tool which is being used to increase the understanding between customers and developers (in the analysis phase). It is also being employed to improve the communication among team members (Nugroho and Chaudron, 2009) and to broaden the understanding of how software works (in both the development and maintenance phases).

Despite all this, any type of investment must be justified from an economic point of view, in the sense that there should be a payback at a later phase. In the context of software projects, therefore, investment in modelling should be justified by benefits that can be gained later, during software development or maintenance. Such benefits
might include improved productivity and better product quality. The existence of these potential advantages is one of the main reasons for investigating whether the use of the UML can generate important differences that make the costs worthwhile. This is particularly true in the context of software maintenance, which consumes a large part of software development resources. Maintenance typically accounts for 40 to 80 percentage of software costs (Glass, 2002; Pressman, 2005).

More than fifteen years have passed since 1997, when the UML was first introduced as a modelling language to describe object-oriented software systems. A comprehensive review of the empirical literature on software engineering is an important step towards its use in maintenance. That being so, it would be useful for the software industry to know what empirical evidence exists as regards the use of the UML in the maintenance of source code and the maintenance of the UML diagrams themselves. With this purpose in mind, we decided to perform a review of the literature related to this issue in order to answer our main research question:

**What is the current existing empirical evidence with regard to the use of UML diagrams in source code maintenance and the maintenance of the UML diagrams themselves?**

The scientific literature found differentiated several types of systematic reviews (Petticrew and Roberts, 2006), including the following:

- Conventional systematic reviews (Petticrew and Roberts, 2006), which aggregated results concerning the effectiveness of a treatment, intervention, or technology, and were related to specific research questions, and
- Mapping studies (Arksey and O’Malley, 2005) whose aim was to identify all research related to a specific topic, i.e. to answer broader questions related to trends in research. Typical questions are exploratory.

This paper aims to present a systematic review of papers dealing with the use of UML diagrams in source code maintenance and the maintenance of the UML diagrams themselves. This work is classified as a secondary study since it is a review of primary studies. A proper systematic review of the literature follows a rigorous and systematic approach, like that described by Kitchenham and Charters in (Kitchenham and Charters, 2007). This approach was therefore used as a basis to perform a systematic mapping study, owing to the need to adopt systematic approaches towards assessing and aggregating research outcomes in order to provide a balanced and objective summary of research evidence for this particular topic. Our goal was to collect evidence that could be used to guide research and practice, and we therefore consider this systematic mapping study to be part of the evidence based software engineering effort (Kitchenham and Charters, 2007).

With regard to our main research question, it is important to note that, on the one hand, most companies only maintain the source code of a system without updating the diagrams which represent it (Forward et al., 2010). This may influence the subsequent maintenance of the same system, which might be misunderstood as a result of inconsistencies. The experience of several researchers belonging to the FaST-RE
research group (Leiden University), headed by Michel R. V. Chaudron, which has a long tradition of collaborating with industrial partners, reflected that the lack of maintenance of diagrams can occur for several reasons such as time constraints, a low level of comprehension of diagrams, or a low modifiability of diagrams.

On the other hand, software development itself is becoming more model-centric (Mohagheghi et al., 2009). Both the OMG Model Driven Architecture (OMG, 2003) and the recent growth of the Model-Driven Development (MDD) software engineering paradigm (Atkinson and Kühne, 2003) emphasize the role of modelling in the development (and hence in the maintenance) of software systems. MDD treats software development as a set of transformations between successive models, from requirements to analysis, to design, to implementation, and to deployment (Thomas, 2004). MDD’s defining characteristic is that the primary focus and products of software development/maintenance are models, rather than computer programs.

These facts also led us to focus on the importance of investigating the maintenance of the models/diagrams themselves, and of attempting to discover whether the diagrams are understandable and modifiable to an extent that would allow maintainers to perform the changes that need to be made to them at the same time as they are maintaining the source code.

At this point, we shall define what maintenance is considered to be in this paper. The software production process can be broken down into various phases. The most commonly-defined phases are the following: analysis and requirements definition, design, implementation, testing and installation, and operation and maintenance (Priestley and Utt, 2000).

Maintenance is defined as the modification of a software product after it has been delivered (to users or customers) in order to correct faults, improve performance or other attributes, add new functionalities, or adapt it to a changing environment (IEEE, 1993).

As is common in the maintenance literature (Genero et al., 2001), we considered two of the major types of tasks included in maintenance:

- Understanding/comprehending the software artifact: in order to modify a program, programmers need to understand its functionality and requirements, its internal structure and its operating requirements. This is necessary if the impact of changes is to be understood.
- Modification of the software artifact: in order to incorporate the necessary changes, a maintenance engineer should create, modify and verify data structures, logic processes, interfaces and documentation. Programmers should have an in-depth knowledge of the impact on the system of the changes being made in order to avoid possible side effects.

The aforementioned tasks used as part of the maintenance and the maintainability sub-characteristics proposed in the ISO 25000 (ISO/IEC, 2008) were additionally used as a basis to focus our study on modifiability as part of maintenance tasks. Moreover, although understandability is not considered to be a maintainability sub-characteristic
in the ISO 25000, we consider it to be part of maintenance since a considerable amount of works judge understandability to be a factor that influences maintainability (Briand et al., 2001a; Deligiannis et al., 2002; Genero et al., 2007; Harrison et al., 2000). A software artefact must be well-understood before any changes can be made to it. We also took misinterpretation as being a factor that influences the understandability of a system and thus its maintainability.

The remainder of this paper is organized as follows. Section 2.2 presents a brief discussion of related work. This is followed by the explanation of each step of the systematic mapping study process. The explanation of the steps involved in planning and conducting the study can be found in Sections 2.3 and 2.4, respectively. The reporting results step and data synthesis of the systematic mapping study are presented in Section 2.5. A discussion of the results is presented in Section 2.6, along with the implications of these findings. Section 2.7 presents the threats to the validity of this systematic mapping study. Finally, in Section 2.8, the conclusions reached are set out and future research possibilities are discussed.

2.2. RELATED WORK

To the best of our knowledge only one research study which presents a literature review on the effect that the use of the UML by developers has on the design and maintenance of object-oriented software (Dzidek, 2008), and this review is different from that performed here. On the one hand, it is worth noting that the process carried out by Dzidek (Dzidek, 2008) is not strictly a systematic literature review or a systematic mapping study. The study relies on the terms that a systematic literature review or systematic mapping study provides, but some activities are missing. In the planning step of the review there are no details about the creation of a comprehensive review protocol. Moreover, the research study limits the search for documents to a number of journals and conferences that the author knows of and considers relevant (i.e., a manual search was performed). The problem with this is that there are also some important conferences on the subject, such as the International Conference on Software Maintenance (ICSM) or the European Conference on Software Maintenance and Reengineering (CSMR), among others, which were not taken into account by the author. Another difference is the search period, since in our study the period under study is broader (from 1997 until 2010) than that in (Dzidek, 2008), which covers the period until 2006. In addition, the research questions were different. Dzidek focuses on efforts to find documents that are relevant to the state-of-the-art of the use of the UML in industry, and also seeks to discover the influence of using supporting tools. In contrast, although the systematic mapping study presented in this paper also aims to obtain the state-of-the-art of empirical studies related to the use of the UML in maintenance tasks, it fixes its attention on how rigorously such empirical studies have been performed (variables, diagrams used, threats to validity, etc.) rather than concentrating on the tools used. The effect of all these differences is that Dzidek obtained 23 primary studies, which is half the number of primary studies found in the systematic mapping study presented in this paper, and only 10 papers are selected as primary studies in both pieces of work.
Another study, that of Budgen (Budgen et al., 2011a), does report a systematic literature review of empirical studies related to the UML, but its focus is different from our systematic mapping study. It aims to discover empirical studies concerning the use of the UML in general, in addition to those concerning some of the properties of the UML. Our focus, however, is specifically on the use of UML diagrams in software maintenance. We do not consider those papers that deal with the properties of the UML as a language, since we have concentrated on factors of UML diagrams when they are used in software maintenance. The aforementioned study, moreover, took as its main task an investigation into the methods and tools used for UML notations or extensions (such as the Object Constraint Language (OCL)). In addition, the papers in it were only those published until 2008, and our study period is therefore broader. It is also important to highlight that its data extraction form concentrates mainly on obtaining data related to how the empirical study was carried out, and on the type of subjects and tasks. The environments in which the studies were carried out, the kind of systems used, or the origin of the diagrams are not considered.

Genero et al. (Genero et al., 2011) present a systematic literature review on the quality of UML diagrams, but it is not focused exclusively on maintenance, as the systematic mapping study reported on in this paper is (less than 20% of the primary studies were related to maintainability of the UML diagrams). It is worth mentioning that although they did not focus solely on empirical evidence, 30% of their primary studies were empirical (24% of the total were experiments, 5% case studies, and 1% surveys).

Despite the fact that UML is widely used in practice, little is known about how UML is actually used. A survey on the use of UML presented by Dobing & Parsons in (Dobing and Parsons, 2006) describes which UML notations (diagram types) are commonly used. These authors analyze the use of each diagram from different points of view, i.e., how are they used for client verification, for programmers’ specifications or for maintenance documentation purposes. This last category, i.e., maintenance documentation, is that which is most closely related to the purpose of the systematic mapping study presented herein. The authors stated that class, sequence, and use case diagrams are most often used in practice.

2.3. PLANNING

Planning includes dividing the workload amongst the researchers and determining how the researchers will interact and conduct the review; it also encompasses the development of the review protocol itself. The planning step is concerned with developing the protocol that prescribes a controlled procedure for conducting the review. Our protocol included objectives, research questions, a search strategy, and inclusion/exclusion criteria (as part of the selection strategy), along with a data extraction form and the quality assessment criteria. The protocol was revised and refined in iterations after the execution of each of the respective activities in the review.
Our main objective is to gather empirical evidence in order to discover whether software maintainers perform maintenance tasks better (in terms of less time and fewer defects) when a UML diagram is available, or whether the use of UML diagrams does not decrease maintainers’ productivity or quality. We focus on maintenance in general, independently of whether the maintenance is performed solely on the code, or also on the diagrams, or only on the diagrams and then translated to code. This objective allowed us to derive a series of questions that we hoped to answer with the results of our research:

RQ1: Which diagrams are most frequently used in studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams?

RQ2: Which dependent variables are investigated in the empirical studies? How are they measured?

RQ3: What is the state-of-the-art in empirical studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams?

RQ4: Which of the factors studied influence the maintainability of a system (source code and diagrams)?

We aim to gather the existing empirical evidence within the area of the maintenance of the UML diagrams or their use in maintenance tasks. In particular, we wish to take into account empirical research on the topic. The latter is particularly important, since it provides information about what we actually know in terms of evidence.

Based on our research questions, we selected the major search terms, which are “UML”, “Maintenance” and “Empirical”. The alternative spellings and synonyms of, or terms related to, the major terms are denominated as alternative terms (and are shown in Table 2.1). The search terms were constructed using the following steps (Brereton et al., 2007):

1. Define major terms.
2. Identify alternative spellings and synonyms of, or terms related to, major terms.
3. Check the keywords in any relevant papers we already have.
4. Use the Boolean OR to incorporate alternative spellings, synonyms or related terms.
5. Use the Boolean AND to link the major terms.

Only “Unified Modelling Language” was considered to be synonymous with UML, rather than adding the name of each UML diagram. This was because we aimed to cover all of the thirteen diagrams that the UML includes.

As explained in the introduction, for terms related to maintenance, we took all the maintainability sub-characteristics proposed in the ISO 25000 (ISO/IEC, 2008). Although understandability is not considered to be a maintainability sub-characteristic in the ISO 25000, we included terms related to understandability since a considerable amount of works judge understandability to be a factor that influences maintainability (Briand et al., 2001a; Deligiannis et al., 2002; Genero et al., 2007; Harrison et al.,
A software artefact must be well-understood before any changes can be made to it. So, we also took misinterpretation to be one of the factors that influence the understandability of a system and thus its maintainability. This being the case, we added this term to the search string.

**Table 2.1. Search string.**

<table>
<thead>
<tr>
<th>Major terms</th>
<th>Alternative terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Maintainability, Modularity, Reusability, Analyzability, Changeability, Evolution, Evolvability, Modification, Stability, Testability, Comprehensibility, Comprehension, Understandability, Understanding, Misinterpretation</td>
</tr>
<tr>
<td>Empirical</td>
<td>Experiment, Survey, Case study, Action research</td>
</tr>
</tbody>
</table>

We performed and automated searches in 6 digital libraries rather than performing a manual search based on the following assumptions (although subsequently published literature (Kitchenham et al., 2010) contradicts one of these assumptions): 1) it saves times during the search; 2) all the sources in digital libraries are correctly indexed, so all the available sources, i.e., conferences and journals, will be taken into account, thus contributing towards improving the completeness of the results; and 3) if the search string is well constructed and it is sufficiently robust, all available research will be found.

The complete search strategy is summarized in Table 2.2.

**Table 2.2. Summary of search strategy.**

| Databases searched | • SCOPUS database  
|                    | • Science@Direct with the subject Computer Science  
|                    | • Wiley InterScience with the subject of Computer Science  
|                    | • IEEEExplore  
|                    | • ACM Digital Library  
|                    | • SPRINGER database |
| Target items       | • Journal papers  
|                    | • Workshop papers  
|                    | • Conference papers |
| Search applied to  | Abstract – when this was not possible we searched in the full text |
| Language           | Papers written in English |
| Publication period | From January 1997 to January 2010 (inclusive) |

The papers that were included were those that presented any kind of empirical study dealing with the use of the UML in maintenance-related tasks, which had been written in English and which were published between 1997 and January 2010. As the UML was adopted by OMG in 1997 (OMG, 1997), it made no sense to search before that period.

The following papers were excluded: pure discussion and opinion papers, studies available only in the form of abstracts or PowerPoint presentations, duplicates (for example, the same paper included in more than one database or in more than one database).
journal), research focusing on issues other than maintenance processes using the UML and their empirical validation, or where the major terms were only mentioned as a general introductory term in the paper’s abstract. Papers were also excluded if they dealt with extensions to the UML, because our interest lay in the UML itself, in the form specified by the OMG.

A summary of the selection strategy is shown in Table 2.3.

| Inclusion criteria | • Only English  
|                    | • Date of publication: from January 1997 to March 2010  
|                    | • Published and refereed works  
|                    | • Terms satisfying the search string  
| Exclusion criteria for titles and abstracts | • Pure discussion and opinion papers, studies available only in the form of abstracts or PowerPoint presentations  
| | • Where the UML or maintenance are mentioned only as general introductory terms in the paper’s abstract and an approach or another type of proposal is among the paper’s contributions  
| Exclusion criteria for full text | • Papers that deal with UML extensions  
| | • Papers that do not contain results of empirical studies  
| | • Papers that are a summary of a workshop  

A template (Table 2.4) for data extraction was produced to ease the activity of synthesizing the data gathered, inspired by the work of (Šmite et al., 2010). Each of the papers was classified into several categories, signifying that the template has two parts, the first of which is related to the metadata of the paper (title, author and name of publication) and the second one of which is related to the classification of the paper according to the following categories:

1. **Year of publication**: The year in which the paper was published. This field is not related to any RQ, but it contributes additional results.
2. **Type of publication**: This could be a journal, a conference or a workshop. This field is not related to any RQ, but it contributes additional results.
3. **Empirical methods**: This could be a case study, survey, experiment, or action research. This field is related to RQ3.
4. **Contexts**: This could be a laboratory or an industrial context. This field is related to RQ3.
5. **Number of subjects**: This represents the sample size, or number of subjects involved in the empirical study. This field is related to RQ3.
6. **Type of subjects**: This could be students, professors, and professionals. This field is related to RQ3.
7. **Dependent variables and their measures**: the dependent variables selected and the measures used to measure them. This field is related to RQ2.
8. **Independent variables**: the independent variables selected in the study, i.e., what the treatments compared were. This field is related to RQ3.
Table 2.4. Data extraction form.

<table>
<thead>
<tr>
<th>Title</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source (name of the journal/conference)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Year of publication</td>
<td>Conference</td>
<td>Journal</td>
<td>Workshop</td>
</tr>
<tr>
<td>Type of publication</td>
<td>Survey</td>
<td>Case study</td>
<td>Action Research</td>
</tr>
<tr>
<td>Empirical method</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td>Industry</td>
<td>Laboratory</td>
<td></td>
</tr>
<tr>
<td>Number of subjects</td>
<td>Students</td>
<td>Academic staff</td>
<td>Professionals</td>
</tr>
<tr>
<td>Type of subjects</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependent variables/Measures</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent variable</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tasks (number and duration)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Available diagrams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object to maintain</td>
<td>Code</td>
<td>Diagrams</td>
<td>Code + Diagrams</td>
</tr>
<tr>
<td>Type of system</td>
<td>Synthetic</td>
<td></td>
<td>Real</td>
</tr>
<tr>
<td>Origin of diagrams</td>
<td>Reverse Engineering</td>
<td>Development process</td>
<td></td>
</tr>
<tr>
<td>Summary/Comments</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

9. **Tasks**: the type of task performed during the empirical studies (test of the understandability of a diagram through a questionnaire, modification tasks, etc.) and its duration (expressed in minutes). This field is related to RQ2.

10. **Available diagrams**: In the original version of the UML submitted in 1997, there were 9 different diagrams with which to model systems from different viewpoints. In UML 2.0 there are 4 new diagrams, making a total of 13. One of these, the communication diagram, has a different name to that of the original UML collaboration diagram. We use the name from the original version, as it is seen more frequently. This field is related to RQ1.

11. **Objects to maintain**: the empirical study can deal with maintenance tasks in the code, in the code and diagrams or in the diagrams only. This field is related to RQ3.
12. **Type of system**: the diagrams could represent a real system or a prototype created specifically for the experiment, which we have called a synthetic system. This field is related to RQ3.

13. **Origin of diagrams**: this could be reverse engineering or a development process. This field is not related to any RQ, but it contributes additional results.

14. **Summary/Comments**: a brief description of what is done in the paper, and which factors were studied. This field is related to RQ4.

As quality criteria for the primary study selection we decided to include those papers that have been published in refereed sources and that also contain empirical data. In addition, as is suggested in (Kitchenham and Charters, 2007), a quality checklist was defined for data synthesis and analysis. As the research field is still immature, and since there are no other review papers on the same topic, we did not want to exclude papers. The quality assessment was to be performed once the primary studies had been selected, the purpose being to assess the rigor of each empirical study. We planned to verify whether or not the publications did indeed either mention or discuss issues related to each of the quality metrics.

The criteria used for quality assessment were based on 8 questions (see Table 2.5) which were extracted from previous work, such as the quality criteria presented by Dybå and Dingsøyr (Dybå and Dingsøyr, 2008a, 2008b), who based their quality assessment criteria on the Critical Appraisal Skills Programme (CASP) (CASP UK, 2008) and principles of good practice for conducting empirical research in software engineering (Kitchenham et al., 2002). Only a few minor changes were made in order to customize the detailed sub-criteria presented in Appendix B of (Dybå and Dingsøyr, 2008a) to our study. A summary of the quality assessment criteria is presented in Table 2.5, along with the way in which each criterion was scored.

### 2.4. CONDUCTING THE REVIEW

Before presenting the *conducting* step, we wish to clarify that in the remainder of this document, the term *paper* is used to refer to the articles published in conferences, journals or workshops that have been retrieved and are analysed in this literature review. The term *primary study* (or *empirical study*) is used to refer to an experiment or another type of empirical study reported in each paper. Both original studies and replications are counted as primary studies. As a paper may contain more than one primary study, the total number of primary studies is greater than the total number of papers.

The *conducting* step includes data retrieval, study selection, data extraction, and data synthesis. In this section, the execution of these activities, performed according the protocol defined above, is explained.

Three researchers were involved in the review, which took around 6 months to complete, and a schedule of which is shown in Table 2.6. This illustrates the *planning*, *conducting* and *reporting* steps on a time scale, along with the outcomes obtained as part of each step.
### Table 2.5. Quality checklist.

<table>
<thead>
<tr>
<th>Quality criteria</th>
<th>Quality metrics (Max=40 points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Regarding <em>Aims</em> and <em>Objectives</em></td>
<td><strong>Total: 5 points</strong></td>
</tr>
<tr>
<td>1.1. Is there a clear statement of the aims of the research?</td>
<td>If there is a statement with the aims/objectives of the research +1</td>
</tr>
<tr>
<td>1.2. Is there a rationale for why the study was undertaken?</td>
<td>If there is an explanation of the reason for undertaking the research +1</td>
</tr>
<tr>
<td>1.3. Do the authors state research questions?</td>
<td>If the RQ are presented +1</td>
</tr>
<tr>
<td>1.4. Do the authors state hypotheses and their underlying theories?</td>
<td>If the hypotheses are presented +1 If the hypotheses are explained +1</td>
</tr>
<tr>
<td>2. Regarding the <em>Context</em></td>
<td><strong>Total: 5 points</strong></td>
</tr>
<tr>
<td>2.1. Do the authors describe the sample and experimental units (=experimental materials and participants as individuals or teams)?</td>
<td>If the materials are presented +1 If the participants are presented +1</td>
</tr>
<tr>
<td>2.2. Was the recruitment strategy appropriate to the aims of the research?</td>
<td>If the recruitment strategy is explained +1</td>
</tr>
<tr>
<td>2.3. Do the authors explain how experimental units were defined and selected?</td>
<td>If the selection of materials is presented +1 If the selection of subjects is presented +1</td>
</tr>
<tr>
<td>3. Regarding the <em>Design of the Experiment</em></td>
<td><strong>Total: 2 points</strong></td>
</tr>
<tr>
<td>3.1. Has the researcher justified the research design (e.g., have the authors discussed how they decided which methods to use - blocking, within or between-subject design; do treatments have levels)?</td>
<td>If the design of the experiment is justified +1</td>
</tr>
<tr>
<td>3.2. Do the authors define/describe all treatments and all controls?</td>
<td>If there is an explanation of the treatments +1</td>
</tr>
<tr>
<td>4. Regarding <em>Control Group</em></td>
<td><strong>Total: 1 point</strong></td>
</tr>
<tr>
<td>4.1. Was there a control group with which to compare treatments?</td>
<td>If there is a control group +1</td>
</tr>
<tr>
<td>5. Regarding <em>Data Collection</em></td>
<td><strong>Total: 5 points</strong></td>
</tr>
<tr>
<td>5.1. Are all measures clearly defined (e.g., scale, unit, counting rules)?</td>
<td>If the measures are defined +1</td>
</tr>
<tr>
<td>5.2. Is it clear how data was collected (e.g., semi-structured interviews, focus group, etc.)?</td>
<td>If there is an explanation of how the data were collected +1</td>
</tr>
<tr>
<td>5.3. Is the form of the data clear (e.g., tape recording, video material, notes, etc.)?</td>
<td>If there is an explanation of the kind of data +1</td>
</tr>
<tr>
<td>5.4. Are the tasks clearly defined (multiple choice, open questions, etc.)?</td>
<td>If there is an explanation of the tasks +1</td>
</tr>
<tr>
<td>5.5. Are quality control methods used to ensure consistency, completeness and accuracy of collected data?</td>
<td>If the control methods are explained +1</td>
</tr>
<tr>
<td>6. Regarding <em>Data Analysis Procedures</em></td>
<td><strong>Total: 10 points</strong></td>
</tr>
<tr>
<td>6.1. Do the authors justify their choice/describe the procedures/provide references to descriptions of the procedures?</td>
<td>If there is a justification of the choice +1 If there is description of procedures +1 If there are references to procedures +1</td>
</tr>
<tr>
<td>6.2. Do the authors report significance levels, effect sizes and power of tests?</td>
<td>If there is a significance level +1 If there is an effect size +1</td>
</tr>
</tbody>
</table>
### Quality criteria

<table>
<thead>
<tr>
<th>Quality criteria</th>
<th>Quality metrics (Max=40points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.3. If outliers are mentioned and excluded from the analysis, is this justified?</td>
<td>If there is a power of test +1</td>
</tr>
<tr>
<td>6.4. Has sufficient data been presented to support the findings?</td>
<td>If there is an explanation of outliers +1</td>
</tr>
<tr>
<td>6.5. Do the authors report or provide references to raw data and/or descriptive statistics?</td>
<td>If there is sufficient data +1 If there is a link to raw data +1 If there are descriptive statistics +1</td>
</tr>
<tr>
<td>7. Regarding Threats to Validity / Bias</td>
<td><strong>Total: 7 points</strong></td>
</tr>
<tr>
<td>7.1. Has the relationship between researchers and participants been adequately considered?</td>
<td>If the relationship has been considered +1</td>
</tr>
<tr>
<td>7.2. If the authors were the developers of some or all of the treatments, do the authors discuss the implications of this anywhere in the paper?</td>
<td>If the implications were discussed +1</td>
</tr>
<tr>
<td>7.3. Was there random allocation to treatments?</td>
<td>If allocation is random +1</td>
</tr>
<tr>
<td>7.4. Was training and conduct equivalent for all treatment groups?</td>
<td>If training is equivalent +1</td>
</tr>
<tr>
<td>7.5. Was there allocation concealment, i.e. did the researchers know to which treatment each subject was assigned?</td>
<td>If researcher doesn’t know which treatment is received by each subject (double blind) +1</td>
</tr>
<tr>
<td>7.6. Do the researchers discuss the threats to validity?</td>
<td>Threats to validity are explained +1</td>
</tr>
<tr>
<td>8. Regarding Conclusions</td>
<td><strong>Total: 5 points</strong></td>
</tr>
<tr>
<td>8.1. Do the authors present results clearly?</td>
<td>If results are clear +1</td>
</tr>
<tr>
<td>8.2. Do the authors present conclusions clearly?</td>
<td>If conclusions are clear +1</td>
</tr>
<tr>
<td>8.3. Are the conclusions warranted by the results and are the connections between the results and conclusions presented clearly?</td>
<td>If conclusions are extracted from the results +1</td>
</tr>
<tr>
<td>8.4. Do the authors discuss their conclusions in relation to the original research questions?</td>
<td>If there is a link between RQ and conclusions +1</td>
</tr>
<tr>
<td>8.5. Do the authors discuss whether or how the findings can be transferred to other populations, or consider other ways in which the research can be used?</td>
<td>If there is a value for research or practice +1</td>
</tr>
</tbody>
</table>

In the *planning* step the protocol was defined, the details of which were explained in the previous section. In the *conducting* step we can see how the different documents were selected according to their relevance. The outcomes show the results after each step, such as the protocol review, or the number of papers that we had at a given time.

The protocol was developed by the three authors of this paper, and the searches were then carried out by the first author of the paper. The results of these were used by the second author to perform the first study selection, using abstracts and titles. The first author of the paper removed any duplicates and retrieved the data needed in the candidates for primary studies. After this first cycle, the protocol was improved by the three researchers. The study of the selected papers, along with their classification based on the full text, was carried out by the first author of the paper, who resolved any queries she had with the other two authors. A sample of 10 random papers was selected by the second and third authors of the paper to check the classification performed by the first author, and all three of us agreed on the classification of the sample.
Table 2.6. Systematic Mapping Study outline.

<table>
<thead>
<tr>
<th>Chronology</th>
<th>Planning</th>
<th>Conducting</th>
<th>Reporting</th>
<th>Outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>March 2010</td>
<td>Protocol development</td>
<td></td>
<td></td>
<td>Review protocol</td>
</tr>
<tr>
<td>April 2010</td>
<td></td>
<td>Data retrieval</td>
<td></td>
<td>Table with the metadata of the papers (808).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Study selection on basis of abstracts and titles</td>
<td></td>
<td>Table with the metadata of the primary studies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>selected (148).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Remove duplicates</td>
<td></td>
<td>Table with the metadata of the papers (85)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Retrieval of the files of the primary studies</td>
<td></td>
<td>Repository of primary studies</td>
</tr>
<tr>
<td>May 2010</td>
<td>Protocol improvement</td>
<td>Pilot data extraction</td>
<td></td>
<td>Data extraction form with the classification</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Study selection, quality assessment, and</td>
<td>scheme refined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>classification based on the full text</td>
<td>85 primary studies reviewed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Resolution of queries in classification of</td>
<td>Data extraction form completed with the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>primary studies in group</td>
<td>classification of papers (53).</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Data synthesis</td>
<td>Revised classification of the primary studies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(38).</td>
</tr>
<tr>
<td>September 2010</td>
<td></td>
<td></td>
<td></td>
<td>Final report</td>
</tr>
</tbody>
</table>

The Planning for the systematic mapping study began in March 2010, and papers published between 1997 and March 2010 were retrieved in April 2010. 808 papers were found (Figure 2.1). The title and abstract of each of the papers was examined and all those not dealing with empirical studies concerning the use of the UML in maintenance tasks were excluded, thus reducing the total to 148 papers. 63 duplicate papers (since there is some overlap between the electronic databases covered by the
different search engines, and some papers were therefore found by more than one of them) were discarded. The inclusion and exclusion criteria were then applied by reading the full text of each of the 85 remaining papers, leading to the discarding of 32 other papers.

We then detected that some of the empirical studies were included in more than one paper, so we also eliminated those 15 papers which contained results of empirical studies that had been summarized in other papers (we maintained the last published paper related to the same empirical study, which contained more details owing to the fact that they were journal papers). The final classifications were made on the final 38 papers, which reported 66 primary studies (empirical studies), and these were then analyzed and the results interpreted.

![Figure 2.1. Selection process.](image)

We were conscious that the search string was extremely long, and observed that, owing to the limitation of the search engines, such a long string could not be used directly. It was therefore necessary to tailor the search string to each digital library by splitting the original and then combining the results manually. Current search engines are not designed to support systematic literature reviews or systematic mapping studies. Unlike medical researchers, software engineering researchers need to perform
resource-dependent searches (Brereton et al., 2007). In order to alleviate, in part, some of the limitations of the search engines, we have used the tool known as SLR-Tool (Fernández-Sáez et al., 2010), which allowed us to refine the searches. More information on how the original search string was tailored to each digital library is shown in APPENDIX C. THE SEARCH STRINGS (related to Chapter 2)

2.5. REPORTING RESULTS AND DATA SYNTHESIS

Finally, the reporting step presents and interprets the results. In this section, we present the results of the systematic mapping study based on the 38 papers eventually selected. The structure of the results is based on the research questions which were set out above in Section 2.3. Data extracted from the papers reviewed were analyzed both quantitatively and qualitatively in order to answer the research questions.

2.5.1. Counting empirical studies

It is first necessary to explain some details about how the papers and the empirical studies were counted. Some papers collected in our investigation reported a single empirical study or a single replication. However, some other papers reported more than one experiment or replication in a single paper and others reported one or more replications together with an original study in a single paper. In these cases, for only one paper we counted each replication and original separately (i.e., we had more than one primary study), and this is why there are a different number of papers and empirical studies. In the remainder of this paper each paper will be called $P_x$, in which $x$ is the number of the paper. Each $P_x$ is a complete reference to a paper listed in APPENDIX A. LIST OF PRIMARY STUDIES (related to Chapter 2)

We started with 38 papers and obtained a total of 66 empirical studies, since some papers contained the results of more than one empirical study, as explained previously, and it is for this reason that we refer to “empirical studies” rather than “papers” when answering some of the research questions. Moreover, an empirical study may be related to more than one item from each of the categories defined, such as a paper related to the maintenance of class diagrams but also of sequence diagrams. In this case that primary study will therefore be counted twice. On the basis on this, the total number of empirical studies which appear in the result tables may therefore be greater than 66 (resulting in a “fictitious” total). The percentages are consequently calculated by using the “fictitious” total as the basis, rather than by taking the actual number of empirical studies (66) as a starting point. This is done to prevent percentages above 100%, which can sometimes hinder the understandability of the results.

2.5.2. Answers to the research questions

Firstly, we would like to remark that we found that only 2 papers ([P5] and [P8]) of the 38 are related to the use of UML diagrams in the maintenance of source code, and that the rest are related to the maintenance of the UML diagrams themselves. These two papers represent 3 of the 66 empirical studies mentioned above. Although they are the only papers related to the maintenance of source code and they are expected to
answer only the first part of our main research question, they also show results related to the maintenance of the UML diagrams themselves. As such, they also helped us to answer the second part of our main research question (i.e., in the following subsections they will be counted in the results for both parts of the question). In the following subsections we answer our research questions.

2.5.2.1. **RQ1: Which diagrams are most frequently used in studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams?**

We analysed the studies in an attempt to find any reference to the 13 diagrams of the UML 2.3. The results which answer RQ1 are shown in Table 6, although those diagrams of which no evidence was found are omitted. 10.10% of the studies have been classified as not being related to a specific type of UML diagram. The type of diagram that is most frequently studied is the class diagram, in 34.34% of the studies. 17.17% refer to sequence diagrams, 16.16% to statechart diagrams, and 11.11% refer to collaboration diagrams. Only 8.08% of the studies selected relate to use case diagrams and 2.02% to activity diagrams, and only one study focused on deployment diagrams. No studies addressing any of the four new diagrams that were introduced with the UML 2.0 were found.

The low proportion of studies relating to use case diagrams is noticeable (Table 2.7). This may be explained by the fact that there are no studies addressing this type of diagram which are directly related to maintenance tasks. This low figure might also be

<table>
<thead>
<tr>
<th>Available diagram</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class diagrams</td>
<td>34</td>
<td>34.34%</td>
<td>[P3], [P5], [P8], [P9], [P12], [P13], [P14], [P18], [P19], [P20], [P22], [P24], [P26], [P28], [P29], [P32], [P35], [P36], [P37]</td>
</tr>
<tr>
<td>Sequence diagrams</td>
<td>17</td>
<td>17.17%</td>
<td>[P1], [P4], [P5], [P6], [P8], [P10], [P14], [P18], [P21], [P22], [P31], [P32], [P35], [P38]</td>
</tr>
<tr>
<td>Statechart diagrams</td>
<td>16</td>
<td>16.16%</td>
<td>[P1], [P4], [P7], [P21], [P22]</td>
</tr>
<tr>
<td>Collaboration diagrams</td>
<td>11</td>
<td>11.11%</td>
<td>[P1], [P4], [P14], [P21], [P22], [P29], [P34], [P35], [P38]</td>
</tr>
<tr>
<td>UML diagrams</td>
<td>10</td>
<td>10.10%</td>
<td>[P16], [P17], [P23], [P25], [P27], [P30]</td>
</tr>
<tr>
<td>Use case diagrams</td>
<td>8</td>
<td>8.08%</td>
<td>[P2], [P5], [P8], [P14], [P18], [P35]</td>
</tr>
<tr>
<td>Activity diagrams</td>
<td>2</td>
<td>2.02%</td>
<td>[P33]</td>
</tr>
<tr>
<td>Deployment</td>
<td>1</td>
<td>1.01%</td>
<td>[P35]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>99</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
related to the origin of the diagrams. In some cases the diagrams are obtained from the source code, using reverse engineering, and in this case neither use case diagrams nor sequence diagrams are generally available when using open source tools. Furthermore, use cases say nothing about the structure of the system, and hence do not contain information that a maintainer needs to perform changes/modifications.

If we focus on the empirical studies that are related solely to the use of UML diagrams in the maintenance of code, all of these (3) used use case, class and sequence diagrams.

As mentioned above, the UML diagram that is studied most extensively is the class diagram, as seen in the results presented in (Dobing and Parsons, 2006), in which the most widely-used UML diagram in maintenance documentation is the class diagram. In (Dobing and Parsons, 2006) other rankings are provided based on different points of view or key purpose as is mentioned in that paper. We focus solely on the ranking related to maintenance documentation because it is the ranking which is most frequently related to that presented here. Moreover, our study places the sequence and statechart diagrams in high positions of use, which is consistent with the results provided in (Dobing and Parsons, 2006).

2.5.2.2. RQ2: Which dependent variables are investigated in the empirical studies? How are they measured?

The variables investigated when the maintainability of the UML diagrams is studied are now shown, ordered by the type of diagram to which each is related: class diagrams (see Table 2.8), statechart diagrams (see Table 2.9), sequence diagrams (see Table 2.10), collaboration diagrams (see Table 2.11), use case diagrams (see Table 2.13), and activity diagrams (see Table 2.12). We also have another broader category: variables related to UML diagrams in general (see Table 2.14).

If these tables are observed it will be noted that the variable which is most widely-studied is the understandability of the class diagrams (22.64%), followed by the understandability of statechart diagrams (13.21%). Other common dependent variables are the modifiability (12.26%) and the analyzability (7.55%) of class diagrams, since these are considered to be sub-characteristics of maintainability. Maintainability is also studied as a whole (in class diagrams (0.94%) and in the whole system (2.83%)). In addition, there are several studies whose experiments address the understandability of other specific UML diagrams (11.32% of sequence diagrams and 9.43% of collaboration diagrams).
### Table 2.8. Variables and measures for class diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Understandability</strong></td>
<td>time</td>
<td>16</td>
<td>15.09%</td>
<td>[P3], [P9], [P12], [P17], [P18], [P19], [P28], [P29], [P32], [P36], [P37]</td>
</tr>
<tr>
<td></td>
<td>correctness</td>
<td>8</td>
<td>7.55%</td>
<td>[P9], [P18], [P19], [P20]</td>
</tr>
<tr>
<td></td>
<td>F-measure</td>
<td>4</td>
<td>3.77%</td>
<td>[P26]</td>
</tr>
<tr>
<td></td>
<td>accuracy</td>
<td>5</td>
<td>4.72%</td>
<td>[P17], [P24], [P29], [P36], [P37]</td>
</tr>
<tr>
<td></td>
<td>effectiveness</td>
<td>6</td>
<td>5.66%</td>
<td>[P19], [P32]</td>
</tr>
<tr>
<td></td>
<td>errors</td>
<td>2</td>
<td>1.89%</td>
<td>[P9], [P28]</td>
</tr>
<tr>
<td></td>
<td>efficiency</td>
<td>2</td>
<td>1.89%</td>
<td>[P20], [P32]</td>
</tr>
<tr>
<td></td>
<td>perceived</td>
<td>2</td>
<td>1.89%</td>
<td>[P3], [P12]</td>
</tr>
<tr>
<td></td>
<td>comprehensibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>relative time</td>
<td>1</td>
<td>0.94%</td>
<td>[P29]</td>
</tr>
<tr>
<td><strong>Modifiability</strong></td>
<td>time</td>
<td>11</td>
<td>10.38%</td>
<td>[P3], [P11], [P12], [P13], [P19]</td>
</tr>
<tr>
<td></td>
<td>correctness</td>
<td>7</td>
<td>6.60%</td>
<td>[P13], [P19]</td>
</tr>
<tr>
<td></td>
<td>effectiveness</td>
<td>7</td>
<td>6.60%</td>
<td>[P13], [P19]</td>
</tr>
<tr>
<td></td>
<td>perceived</td>
<td>2</td>
<td>1.89%</td>
<td>[P3], [P12]</td>
</tr>
<tr>
<td></td>
<td>comprehensibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>relative time</td>
<td>1</td>
<td>0.94%</td>
<td>[P29]</td>
</tr>
<tr>
<td><strong>Analyzeability</strong></td>
<td>time</td>
<td>6</td>
<td>5.66%</td>
<td>[P3], [P11], [P12], [P13]</td>
</tr>
<tr>
<td></td>
<td>perceived</td>
<td>2</td>
<td>1.89%</td>
<td>[P3], [P12]</td>
</tr>
<tr>
<td></td>
<td>comprehensibility</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>correctness</td>
<td>2</td>
<td>1.89%</td>
<td>[P13]</td>
</tr>
<tr>
<td></td>
<td>effectiveness</td>
<td>2</td>
<td>1.89%</td>
<td>[P13]</td>
</tr>
<tr>
<td><strong>Maintainability</strong></td>
<td>errors</td>
<td>1</td>
<td>0.94%</td>
<td>[P27]</td>
</tr>
<tr>
<td><strong>Quality</strong></td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P21]</td>
</tr>
<tr>
<td><strong>Easy of construct</strong></td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P21]</td>
</tr>
</tbody>
</table>
Table 2.9. Variables and measures for statechart diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability 14 studies 13.21%</td>
<td>time</td>
<td>6</td>
<td>5.66%</td>
<td>[P4], [P7], [P21]</td>
</tr>
<tr>
<td></td>
<td>efficiency</td>
<td>5</td>
<td>4.72%</td>
<td>[P7]</td>
</tr>
<tr>
<td></td>
<td>effectiveness</td>
<td>5</td>
<td>4.72%</td>
<td>[P7]</td>
</tr>
<tr>
<td></td>
<td>F-measure</td>
<td>6</td>
<td>5.66%</td>
<td>[P1], [P7]</td>
</tr>
<tr>
<td></td>
<td>accuracy</td>
<td>3</td>
<td>2.85%</td>
<td>[P4], [P21]</td>
</tr>
</tbody>
</table>

Table 2.10. Variables and measures for sequence diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability 12 studies 11.32%</td>
<td>accuracy</td>
<td>8</td>
<td>7.55%</td>
<td>[P4], [P6], [P14], [P21], [P22], [P31]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>8</td>
<td>7.55%</td>
<td>[P4], [P14], [P18], [P21], [P22], [P38]</td>
</tr>
<tr>
<td></td>
<td>correctness</td>
<td>2</td>
<td>1.89%</td>
<td>[P10], [P18]</td>
</tr>
<tr>
<td></td>
<td>efficiency</td>
<td>1</td>
<td>0.94%</td>
<td>[P10]</td>
</tr>
<tr>
<td></td>
<td>perceived comprehensibility</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>F-measure</td>
<td>1</td>
<td>0.94%</td>
<td>[P1]</td>
</tr>
<tr>
<td>Quality of construction 1 study 0.94%</td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>perceived ease of construction</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
</tbody>
</table>

Table 2.11. Variables and measures for collaboration diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality of construction 1 study 0.94%</td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>Perceived ease of construction</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td>Understandability 10 studies 9.43%</td>
<td>accuracy</td>
<td>8</td>
<td>7.55%</td>
<td>[P4], [P14], [P21], [P22], [P29], [P34]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>9</td>
<td>8.49%</td>
<td>[P4], [P14], [P21], [P22], [P29], [P34], [P38]</td>
</tr>
<tr>
<td></td>
<td>relative time</td>
<td>1</td>
<td>0.94%</td>
<td>[P29]</td>
</tr>
<tr>
<td></td>
<td>perceived comprehensibility</td>
<td>1</td>
<td>0.94%</td>
<td>[P14]</td>
</tr>
<tr>
<td></td>
<td>F-measure</td>
<td>1</td>
<td>0.94%</td>
<td>[P1]</td>
</tr>
</tbody>
</table>
Table 2.12. Variables and measures for activity diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>errors</td>
<td>2</td>
<td>1.89%</td>
<td>[P33]</td>
</tr>
<tr>
<td></td>
<td>number of elements</td>
<td>1</td>
<td>0.94%</td>
<td>[P33]</td>
</tr>
</tbody>
</table>

Table 2.13. Variables and measures for use case diagrams.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability</td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P2]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>1</td>
<td>0.94%</td>
<td>[P18]</td>
</tr>
<tr>
<td>retention</td>
<td>correctness</td>
<td>1</td>
<td>0.94%</td>
<td>[P18]</td>
</tr>
<tr>
<td>Problem-solving</td>
<td>accuracy</td>
<td>1</td>
<td>0.94%</td>
<td>[P2]</td>
</tr>
</tbody>
</table>

Table 2.14. Variables and measures for the UML diagrams, in general.

<table>
<thead>
<tr>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understandability</td>
<td>accuracy</td>
<td>3</td>
<td>2.85%</td>
<td>[P17], [P23]</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>3</td>
<td>2.85%</td>
<td>[P17], [P23]</td>
</tr>
<tr>
<td></td>
<td>errors</td>
<td>2</td>
<td>1.89%</td>
<td>[P16]</td>
</tr>
<tr>
<td></td>
<td>efficiency</td>
<td>1</td>
<td>0.94%</td>
<td>[P30]</td>
</tr>
<tr>
<td></td>
<td>F-measure</td>
<td>1</td>
<td>0.94%</td>
<td>[P25]</td>
</tr>
<tr>
<td>Error detection</td>
<td>errors</td>
<td>2</td>
<td>1.89%</td>
<td>[P16]</td>
</tr>
<tr>
<td>Learnability of</td>
<td>errors</td>
<td>1</td>
<td>0.94%</td>
<td>[P15]</td>
</tr>
<tr>
<td>modelled concepts</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The variables for those empirical studies related to the maintenance of the source code when using UML diagrams (please recall that these are also related to the maintenance of only the UML diagrams themselves) are presented in Table 2.15, and will be denominated as variables related to the system in general (source code and
Most of them are related to the time spent when performing maintenance tasks and the correctness of the solution (including its quality, measured through the number of errors).

### Table 2.15. Variables and measures for the system (source code and diagrams).

<table>
<thead>
<tr>
<th>SYSTEM</th>
<th>Dependent variable</th>
<th>Measure</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Maintainability</td>
<td>time</td>
<td>3</td>
<td>2.85%</td>
<td>[P5], [P8]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>correctness</td>
<td>3</td>
<td>2.85%</td>
<td>[P5], [P8]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>errors</td>
<td>3</td>
<td>2.85%</td>
<td>[P5], [P8]</td>
</tr>
<tr>
<td></td>
<td>Understandability</td>
<td>F-measure</td>
<td>1</td>
<td>0.94%</td>
<td>[P35]</td>
</tr>
</tbody>
</table>

The variety of measures of dependent variables included in the 66 empirical studies presented in the 38 papers included in this systematic mapping study is relatively wide (Table 2.8-Table 2.15). On the one hand, there are several measures that have different names but measure the same concept (for example, in some papers the percentage of correct answers is called correctness, while other papers call this measure effectiveness). In order to construct Table 2.8-Table 2.15 and to count how many papers use the same measure, we have grouped those measures that look at the same concept under one name, so that the resulting number makes sense. The groups of measures with the same name, along with the definition of each measure and the papers in which they have been defined and used, are detailed in APPENDIX B.

As will be noted, most of the measures are based on objective measures, such as the number of correct answers, the number of questions, or the time spent on the tasks, in addition to different calculations based on all of these. On the other hand, a minority of studies use subjective variables, related to the subjects’ perceptions of the variable measured.

We were surprised to discover that none of the studies considered investigated the use of the UML in productivity in software maintenance, since productivity is often a crucial factor which all software development organizations attempt to maximize. One of the reasons for this might be that measuring the impact of using the UML on productivity in a project is no trivial task: it can be both expensive and difficult, although there were two studies, reported in [P5] and [P8], which did investigate the impact of the UML on software maintenance in an experimental setting. We would like to stress that productivity is an indirect measure which needs some recognition and requires some form of model for its derivation, perhaps based on other direct measures but also by using some environmentally dependent factors. Productivity can be derived from various final measures, which could make a comparison difficult. Another explanation for the lack of studies on the use of the UML in productivity in software maintenance might be that UML diagrams are rarely consulted in maintenance tasks. A poor diagram/code correspondence could explain why the UML
diagrams for maintenance are ignored. But it is also very likely that UML diagrams are consulted or not irrespective of their high or low correspondence to the code, as is explained in (Nugroho and Chaudron, 2008).

For each study it is important to know what kind of tasks the subjects had to perform in order to understand why one dependent variable is used rather than another. Most of the studies that were found perform tasks to test the comprehension of the diagrams. This means that most of the tasks which are performed by the subjects involve answering questionnaires. There are also some studies in which the tasks to be carried out are those of modifying a diagram so that it meets certain requirements.

Table 2.16. Results per duration.

<table>
<thead>
<tr>
<th>Duration (in minutes)</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-60</td>
<td>6</td>
<td>9.09%</td>
<td>[P1], [P2], [P15], [P19], [P30], [P38]</td>
</tr>
<tr>
<td>61-120</td>
<td>12</td>
<td>18.18%</td>
<td>[P10], [P13], [P14], [P18], [P20], [P23], [P29], [P33]</td>
</tr>
<tr>
<td>121-300</td>
<td>8</td>
<td>12.12%</td>
<td>[P16], [P24], [P26], [P28], [P31]</td>
</tr>
<tr>
<td>301-1000</td>
<td>2</td>
<td>3.03%</td>
<td>[P5]</td>
</tr>
<tr>
<td>&quot;+&quot; 1000</td>
<td>8</td>
<td>12.12%</td>
<td>[P3], [P8], [P11], [P12], [P13]</td>
</tr>
<tr>
<td>Not specified</td>
<td>26</td>
<td>39.36%</td>
<td>[P4], [P6], [P7], [P9], [P16], [P17], [P19], [P21], [P22], [P25], [P27], [P32], [P35]</td>
</tr>
<tr>
<td>Others</td>
<td>4</td>
<td>6.06%</td>
<td>[P4], [P33], [P34], [P36], [P37], [P38]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>66</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is important to note that in many studies (almost 40%) the duration of the tasks in not specified, and if it is specified, the duration of the task is usually short, from 1 to 2 hours in length (Table 2.16) to avoid the situation of subjects becoming tired and fatigued; this fatigue would be a threat to the internal validity of the studies. In contrast, there are some uncontrolled experiments in which the presence of the supervisor is not necessary and the subjects have one week to complete the tasks (we consider these to be experiments of 168 hours, i.e., 7 days multiplied by 24 hours per day). There are studies which indicate the time in a measure that cannot be translated into minutes (for example, papers that measure the time taken to do the experiments in “sessions”, in which we do not know the length of each one). These studies have not been taken into account in our calculations (as unconstrained time studies) and they are included in the “others” category. This signifies that, of all the studies found, the average time taken amounts to 2166.53 minutes (36h, approximately).

2.5.2.3. **RQ3:** What is the state-of-the-art in empirical studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams?

This subsection presents several issues related to the state-of-the-art in empirical studies concerning the maintenance of the UML diagrams themselves, or to the maintenance of source code when using UML diagrams. These are the following: the type of empirical study (i.e., the empirical methods), the kind of context in which the
empirical studies were executed, the kind of participants in the empirical studies (i.e., the subjects), what was maintained during the study (i.e., the object maintained), the type of systems used during the studies, the treatments of the studies (i.e., the independent variables), and finally, the quality of the empirical studies and papers.

There are many research methods to choose from when carrying out any investigation. We focused only on those studies that are carried out empirically, as dictated by one of the inclusion criteria. The results of the validation classification method are shown in Table 2.17. We would remind the reader that the number of empirical studies is higher than 38 owing to the fact that several papers fall into more than one category, and we therefore have 66 empirical studies (for example, one paper contains both an experiment and a case study). 95.45% of the studies report the results of a controlled experiment, as is shown in Table 2.17 (note that all of the empirical studies concerning the use of the UML in the maintenance of source code are in this category). This finding shows the need to conduct more case studies, as this is a kind of experimentation that deals with real environments and real projects.

<table>
<thead>
<tr>
<th>Empirical method</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment</td>
<td>63</td>
<td>95.45%</td>
<td>[P1], [P2], [P3], [P4], [P5], [P6], [P7], [P8], [P9], [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P25], [P26], [P27], [P28], [P29], [P30], [P31], [P32], [P33], [P34], [P35], [P36], [P37], [P38]</td>
</tr>
<tr>
<td>Case study</td>
<td>3</td>
<td>4.55%</td>
<td>[P27], [P27], [P35]</td>
</tr>
<tr>
<td>Survey</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
</tr>
<tr>
<td>Action Research</td>
<td>0</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>66</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

According to (Yin, 2002), a “case study is an empirical inquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between the phenomenon and context are not clearly evident”. Bearing this definition in mind, although some studies claimed that a case study was being presented, they were removed, because in actual fact they contained only an example.

The context in which the studies were carried out could be an industrial context or a laboratory (Table 2.18). Most of the studies found regarding the maintainability of UML diagrams (83.33%) are the results of experiments that have been conducted in laboratories within academic environments. In the case of the studies on the maintenance of source code, all of these were performed within a laboratory environment. There are also some papers that present the results of empirical studies in industrial settings, but the percentage of this type of studies is very low (4.55%). Those studies that indicated that the subject under study could do the test at home have been considered as having been carried out within a non-controlled context (12.12%).
Table 2.18. Results per context.

<table>
<thead>
<tr>
<th>Context</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laboratory</td>
<td>55</td>
<td>83.33%</td>
<td>[P1], [P2], [P4], [P5], [P6], [P7], [P8], [9], [P10], [P13], [P14], [P16], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P25], [P26], [P28], [P29], [P30], [P31], [P32], [P33], [P34], [P36], [P37], [P38]</td>
</tr>
<tr>
<td>Non-controlled</td>
<td>8</td>
<td>12.12%</td>
<td>[P3], [P11], [P12], [P13], [P15]</td>
</tr>
<tr>
<td>Industrial</td>
<td>3</td>
<td>4.55%</td>
<td>[P17], [P27], [P35]</td>
</tr>
<tr>
<td>Total</td>
<td>65</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.19. Results per type of subject.

<table>
<thead>
<tr>
<th>Type of subjects</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Students</td>
<td>59</td>
<td>77.63%</td>
<td>[P1], [P2], [P3], [P4], [P5], [P6], [P7], [P9], [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P25], [P26], [P28], [P29], [P30], [P31], [P32], [P33], [P34], [P36], [P37], [P38]</td>
</tr>
<tr>
<td>Practitioners</td>
<td>9</td>
<td>11.84%</td>
<td>[P6], [P7], [P8], [P16], [P17], [P27], [P29], [P35], [P38]</td>
</tr>
<tr>
<td>University Lecturers</td>
<td>8</td>
<td>10.53%</td>
<td>[P3], [P7], [P11], [P12], [P26], [P30]</td>
</tr>
<tr>
<td>Total</td>
<td>76</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The average number of subjects used in the empirical studies in the papers found is 41.19. Table 2.19 shows what types of subjects were used in the empirical studies. The majority of empirical studies (77.63%) tended to be carried out with undergraduate students, in the third, fourth or fifth year of the Computer Science degree. This is not necessarily inappropriate (Budgen et al., 2011a; Kuzniarz et al., 2013), because the UML is intended to support design tasks, and students’ design skills are likely to be similar to those of non-expert professionals. A considerably lower percentage of empirical studies was carried out by members of the university teaching staff (10.53%) or by practitioners (11.84%).

These results show that there is a need to perform more empirical studies with practitioners in order to confirm whether the results obtained with students are also valid with the former type of subjects.

We shall now go on to discuss the type of object(s) that had to be maintained. Software maintenance tasks have always required some changes to be made to the source code (Table 2.20). There is the possibility of using diagrams to maintain the code and of updating these diagrams to reflect the changes (16.67%), or there is the option in which the diagrams are the only elements maintained (83.33%). This second option makes sense when attempting to obtain empirical studies about the
understandability of the diagrams. There are no studies that deal with the maintenance of the code of a system supported by the use of the UML diagrams, but we did find studies in which the UML diagrams are not maintained, which is what appears to occur most often in practice.

Table 2.20. Results per object to maintain.

<table>
<thead>
<tr>
<th>Object to maintain</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagrams</td>
<td>55</td>
<td>83.33%</td>
<td>[P1], [P2], [P3], [P4], [P6], [P7], [P9], [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P25], [P28], [P29], [P30], [P32], [P33], [P36], [P37], [P38]</td>
</tr>
<tr>
<td>Code + Diagrams</td>
<td>11</td>
<td>16.67%</td>
<td>[P5], [P6], [P26], [P27], [P31], [P34], [P35]</td>
</tr>
<tr>
<td>Code</td>
<td>0</td>
<td>0.00%</td>
<td>-</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>66</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Owing to the low percentage of empirical studies that examine the maintenance of both diagram and code, there is no evidence to allow us to really know whether the results obtained in studies with isolated diagrams or isolated code can be generalized to real environments. There is thus a need to carry out more studies of this type, which deal with the maintenance of only the UML diagrams themselves as part of the maintenance of the entire system. The performances of maintainers when using up to date diagrams or older versions of the documentation, e.g., that originate from the design of a system, also need to be compared. In our opinion, the degree of correspondence between diagrams and code could influence some of the maintenance tasks. It is supposed that better results would be obtained when this correspondence is high.

Table 2.21. Results per type of system.

<table>
<thead>
<tr>
<th>Type of system</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synthetic</td>
<td>51</td>
<td>73.91%</td>
<td>[P1], [P2], [P3], [P4], [P5], [P6], [P7], [P9], [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P29], [P31], [P32], [P33], [P34], [P38]</td>
</tr>
<tr>
<td>Real</td>
<td>18</td>
<td>26.06%</td>
<td>[P8], [P17], [P19], [P25], [P26], [P27], [P28], [P30], [P35], [P36], [P37]</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>69</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At this point, we should discuss the origin of the materials used in the studies. Most of the empirical studies that were found (73.91%) used diagrams made from synthetic systems such as prototypes or systems developed specifically for the study (Table 2.21). Only 26.06% of the diagrams used represent real systems in operation. There is
a need to perform more empirical studies with real systems, since most studies address diagrams of small systems, using convenience systems such as a library, ATM, etc., which may not accurately represent the behaviour of large industrial systems.

<table>
<thead>
<tr>
<th>Treatments</th>
<th>Description</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML vs. UML</td>
<td>composite states vs. non-composite states (or different nesting levels)</td>
<td>10</td>
<td>15%</td>
<td>[P7]</td>
</tr>
<tr>
<td></td>
<td>diagrams with stereotypes vs. diagrams without stereotypes</td>
<td>9</td>
<td>14%</td>
<td>[P10], [P25], [P26], [P29], [P31], [P37]</td>
</tr>
<tr>
<td></td>
<td>sequence diagrams vs. collaboration diagrams vs. statecharts</td>
<td>3</td>
<td>5%</td>
<td>[P4], [P21]</td>
</tr>
<tr>
<td></td>
<td>sequence diagrams vs. collaboration diagrams</td>
<td>3</td>
<td>5%</td>
<td>[P14], [P38]</td>
</tr>
<tr>
<td></td>
<td>high Level of Detail vs. Low Level of Detail</td>
<td>1</td>
<td>2%</td>
<td>[P20]</td>
</tr>
<tr>
<td></td>
<td>diagrams with geons vs. diagrams without geons</td>
<td>1</td>
<td>2%</td>
<td>[P15]</td>
</tr>
<tr>
<td></td>
<td>animated diagrams vs. non-animated diagrams</td>
<td>1</td>
<td>2%</td>
<td>[P6]</td>
</tr>
<tr>
<td>Measure $X$ vs. measure $Y$</td>
<td>Values of different measures calculated using the diagrams</td>
<td>14</td>
<td>21%</td>
<td>[P3], [P11], [P12], [P13], [P19], [P28]</td>
</tr>
<tr>
<td>UML vs. other modeling languages</td>
<td>OML vs. UML</td>
<td>2</td>
<td>3%</td>
<td>[P22]</td>
</tr>
<tr>
<td></td>
<td>EPC vs. UML</td>
<td>2</td>
<td>3%</td>
<td>[P33]</td>
</tr>
<tr>
<td></td>
<td>OPM vs. UML</td>
<td>1</td>
<td>2%</td>
<td>[P24]</td>
</tr>
<tr>
<td></td>
<td>UML vs. UML-B</td>
<td>1</td>
<td>2%</td>
<td>[P23]</td>
</tr>
<tr>
<td></td>
<td>UML-B vs. UML + event-B diagrams</td>
<td>1</td>
<td>2%</td>
<td>[P23]</td>
</tr>
<tr>
<td>UML vs. non UML</td>
<td>Presence of UML diagrams vs. absence of UML diagrams</td>
<td>5</td>
<td>8%</td>
<td>[P1], [P5], [P8], [P30]</td>
</tr>
<tr>
<td>Using or not a tool</td>
<td>using a tool (metricViewEvolution) vs. Not using a tool</td>
<td>3</td>
<td>5%</td>
<td>[P17], [P18]</td>
</tr>
<tr>
<td>Layout $X$ vs. layout $Y$</td>
<td>different layouts</td>
<td>2</td>
<td>3%</td>
<td>[P9], [P36]</td>
</tr>
<tr>
<td>Defect $X$ vs. defect $Y$</td>
<td>Presence of different kinds of defects in the UML diagrams</td>
<td>2</td>
<td>3%</td>
<td>[P16]</td>
</tr>
<tr>
<td>Notation $X$ vs. notation $Y$</td>
<td>different notations (on the same diagram)</td>
<td>1</td>
<td>2%</td>
<td>[P34]</td>
</tr>
<tr>
<td>Diagrams vs. text</td>
<td>use case diagrams vs. text cases</td>
<td>1</td>
<td>2%</td>
<td>[P2]</td>
</tr>
</tbody>
</table>
### Treatments

<table>
<thead>
<tr>
<th>Treatments</th>
<th>Description</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward diagrams vs. RE diagrams</td>
<td>Forward-designed diagrams vs. Reverse-Engineered diagrams</td>
<td>1</td>
<td>2%</td>
<td>[P27]</td>
</tr>
<tr>
<td>Retrieval method X vs. retrieval method Y</td>
<td>Different retrieval methods of the UML diagrams</td>
<td>1</td>
<td>2%</td>
<td>[P35]</td>
</tr>
<tr>
<td>Transformation rule X vs. transformation rule Y</td>
<td>Different transformation rules between the UML diagrams</td>
<td>1</td>
<td>2%</td>
<td>[P32]</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>66</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We also extracted some information about the independent variables that were used in the empirical studies, i.e., whose treatments were being used in the various experimental studies. Table 2.22 shows that most of the empirical studies (42%) compare different aspects of the UML diagrams, such as diagrams with stereotypes vs. those without them, different levels of details in the diagrams, etc. This is followed by the values of different metrics which measure some aspects of the UML diagrams, such as complexity, size, etc. (21%), the comparison between the UML and other modelling languages (11%), the presence or absence of the UML diagrams (8%), and so on. All the papers related to the use of the UML diagrams when maintaining the source code are in this last category.

Finally, we present the quality assessment results obtained by applying the quality criteria shown in Table 2.5 to the primary studies. These papers were evaluated to discover whether or not they covered these criteria, and the papers were therefore scored by applying the quality measures shown in Table 2.5. It was possible for a primary study to obtain a maximum of 40 points. Based on that number, we decided to consider three categories: high quality (from 25 to 40 points, i.e. papers with more than 60% of the total points), medium quality (from 16 to 24 points, i.e. papers with 40-60% of the total points), and low quality (from 0 to 15 points, i.e. papers with less than 40% of the total points).

Most of the 38 papers containing primary studies obtained a relatively high score in this quality assessment, as is shown in Table 2.23 (note that all of the papers related to the maintenance of source code when using the UML are in this category).

If we focus on the first quality criterion, i.e., that related to the description of *Aims and Objectives*, we can state that most of the papers obtained a high score because only 1 paper obtained less than 2 points out of 5. The majority of the papers had a good description of the context in which the studies was performed, since 73.68% obtained the maximum score (5) in the criterion concerning the description of the *Context*. This might be owing to the fact that the results of experiments in Software
Engineering cannot be generalized to the whole community and the results are only valid for specific contexts, so they should be commented on in a detailed manner. If we focus on the criterion concerning the description of the Design of the Experiment, almost half of the papers (39.47%) provided a complete description of the design of the paper. The same percentage of papers forgot to justify the research design or the description of the treatments, more or less in the same proportion. It is quite surprising that 21.08% of the papers did not obtain any score in this criterion because they did not describe the design of the experiment. Upon focusing on the next criterion, Control Group, we can state that half of the papers (52.63%) clearly describe the control group used to compare the treatments and the other half do not. If we focus on the Data Collection criterion, we can consider that the majority of papers obtained high scores because only 18.42% obtained less than 3 points out of 5, but almost none of them described whether they used a quality control method to ensure the consistency, completeness and accuracy of the data collected. The next criterion concerns the description of the Data Analysis Procedures, in which most of the papers obtained a medium score (50% of the papers obtained from 4 to 6 point out of a maximum of 10). In this case we cannot state that these papers are particularly good or bad at describing this, but it is important to highlight that none of the primary studies provide references to the raw data used to test the results. One important section in papers is that concerning Threats to Validity/Bias, which is related to our next criterion. The maximum score obtained by papers in this category is 4 out of 7 points. As part of its analysis, we consider it important to note that almost no primary study studied the influence between researchers and participants, or the implications of developing a special system to work on during the research, both of which might influence the validity of the results. We would also like to underline the need to clarify whether or not the review was double blind, i.e., whether the researchers know which treatment is received by each subject, in order not to influence the results when checking their responses. Our last criterion is related to the description of the Conclusions of the study, in which the majority of papers obtained a high score (63.16% obtained 4 out of 5 points).

<table>
<thead>
<tr>
<th>Quality</th>
<th>Number of papers</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>4</td>
<td>10.53%</td>
<td>[P6], [P27], [P32], [P35]</td>
</tr>
<tr>
<td>Medium</td>
<td>14</td>
<td>36.84%</td>
<td>[P1], [P3], [P4], [P7], [P12], [P15], [P17], [P19], [P23], [P28], [P30],[P31], [P34], [P37], [P38]</td>
</tr>
<tr>
<td>High</td>
<td>20</td>
<td>52.63%</td>
<td>[P2], [P5], [P8], [P9], [P10], [P11], [P13], [P14], [P16], [P18], [P20],[P21], [P22], [P24], [P25], [P26], [P29], [P33], [P36]</td>
</tr>
<tr>
<td>Total</td>
<td>38</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.5.2.4. RQ4: Which of the factors studied influence the maintainability of a system (source code and diagrams)?

We have extracted the different factors that can influence the maintainability of systems from the studies analyzed in this systematic mapping study (Figure 2.2). The factors are shown in rounded boxes, and the rectangular boxes contain categories that we have added in order to classify all the factors. A factor that has a positive influence is represented with the symbol plus (+), and the negative influence is indicated with the symbol minus (-). A number related to a further explanation in the following paragraph is shown in brackets. As mentioned previously, it is well known that understandability directly influences maintainability (Briand et al., 2001a; Deligiannis et al., 2002; Genero et al., 2007; Harrison et al., 2000). We therefore assume here that those factors that are related to understandability are also related to maintainability.

The content of Figure 2.2 is explained thus:

- The maintainability of a system is influenced by the maintainability of its source code and its documentation, which can consist of a text or models – UML or non UML models. OPM (non-UML) might be considered to be a better notation but only in the context of modelling the dynamic aspect of Web applications. Other notations that are extensions of UML, like UML-B, have a positive influence on maintainability since they facilitate understanding.
- A maintainer’s skill also affects the maintainability of a system, signifying that when a maintainer has some experience, this will have a positive influence on the maintenance of the system.
- The maintainability of the source code is negatively influenced when the complexity of the system is high, but it is positively influenced by a correct traceability from the diagrams to the source code.
- The maintainability of the UML diagrams is positively influenced by the presence of Reverse Engineered diagrams. These, combined with forward design diagrams, help to detect possible errors in the system, which facilitates its maintenance.
- The maintainability of the UML diagrams is positively influenced by the availability of use case, class, sequence, activity and statechart diagrams, and also whether the diagrams contain stereotypes that detail certain characteristics of their elements. All these characteristics are related to the way in which the model is represented.
- Also related to the representation of models is the fact that the level of detail in the UML diagrams additionally affects the maintainability of systems (source code and diagrams), making it ideal to have a higher level of detail.
- The use of composite states also improves the understandability of the UML statechart diagrams. However, a high nesting level of composite state in the UML statechart diagrams negatively influences the understandability of these diagrams.
- With regard to the way in which models are visualized, the availability of interactive views or animations to improve the diagrams improves the visualization of a UML model, thus improving its maintainability. What is more a proper distribution of the elements (an aesthetic diagram layout) improves the
maintainability of a UML diagram. However, the use of textual use cases has a negative influence on maintainability.

Details of how the classification process of the factors was carried out are set out below. The maintainability of a whole can be considered as the sum of the maintainabilities of each of its parts. In this case, only code and diagrams (as part of the documentation of the system) were considered as part of a system.

The maintainability of the diagrams consists of some characteristics that are directly related to the diagram and others that the reader of the diagram introduces. A diagram can be influenced by its representation (the diagram itself, what is represented) and the way in which it is presented to the reader, i.e., its visualization, or its origin.

The maintainability of the code is influenced by both its own characteristics and the characteristics from the diagrams (since these provide complementary information about the code). In addition, the reader of the code introduces some influential factors.

In the following lines we will explain which paper refers to each factor, referencing the numbers that appear in Figure 2.2:

(1) Positive influence of some diagrams:
   a. Class diagrams: [P5], [P8] and [P22]
   b. Statechart diagrams: [P21] and [P22]
   c. Sequence diagrams: [P1], [P4], [P5], [P8], [P14], [P21], [P22] and [P38]
   d. Activity diagrams: [P33]
   e. Use case diagrams: [P2], [P5] and [P8]

(2) Positive influence of stereotypes: [P10], [P24], [P25], [P29] and [P31].

(3) Negative influence of aggregations as a kind of relationship in class diagrams: [P32].

(4) Influence of composite states: the use of composite states is a positive influence, but if the nesting level is high, the influence is negative [P7].

(5) Positive influence of a high level of detail: [P20].

(6) Positive influence of aesthetic quality or layout: [P9], [P30], [P34], [P36] and [P37].

(7) Positive influence of interactive views or the use of animations: [P6], [P15], [P17] and [P18].

(8) Negative influence of the use of textual use cases: [P2].

(9) Negative influence of the defects in the diagrams: [P16].
Figure 2.2. Factors that influence the maintainability of a system.
(10) Positive influence of traceability from diagrams to code: [P35].
(11) Negative influence of the structural complexity: [P3], [P7], [P11], [P12], [P13], [P19], and [P28].
(12) Positive influence of maintainers’ experience and ability: [P26]
(13) Positive influence of other notations compared to UML in modelling the dynamic aspect of Web applications: [P23].
(15) Positive influence of the presence of diagrams extracted from Reverse Engineering: [P24]

2.5.3. Additional results

The results obtained from the classification of papers are presented here within the “others” category.

Figure 2.3 shows that every year an almost constant number of new publications related to the topic of this study appear. This figure may show that interest in this subject has been growing over time, reaching its highest points in 2009. We should point out that the number of papers in 2010 is small, because the search was only performed until March 2010. Results reveal that there is a mean of almost 5 papers published on this topic per year.

![Year of publication](image)

**Figure 2.3. Number of papers per year.**

When analyzing the types of publication, we found that 39.47% of the papers (15 papers) were published in conferences, 47.37% in journals (18 papers) and 13.16% in workshops (5 papers). The first paper in a journal appeared in 2002, with this figure increasing over the following years to a maximum in 2009, when it reached its highest
level of 5 papers. This coincides with one of the years with the highest number of publications (Figure 2.3). The use of UML diagrams in maintenance tasks has been judged to be a “hot topic”, given the number of publications. The field is nonetheless quite mature, as is demonstrated by the percentage of journal papers.

Table 2.24. Number of papers per type of publication.

<table>
<thead>
<tr>
<th>Journal/Conference</th>
<th>Number of papers</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information and Software Technology</td>
<td>4</td>
<td>10.53%</td>
</tr>
<tr>
<td>IEEE Transactions on Software Engineering</td>
<td>3</td>
<td>7.89%</td>
</tr>
<tr>
<td>Empirical Software Engineering</td>
<td>2</td>
<td>5.26%</td>
</tr>
<tr>
<td>International Conference on Program Comprehension</td>
<td>2</td>
<td>5.26%</td>
</tr>
<tr>
<td>International Symposium on Empirical Software Engineering</td>
<td>2</td>
<td>5.26%</td>
</tr>
<tr>
<td>International Symposium on Empirical Software Engineering and Measurement</td>
<td>2</td>
<td>5.26%</td>
</tr>
<tr>
<td>International Workshop on Visualizing Software for Understanding and Analysis</td>
<td>2</td>
<td>5.26%</td>
</tr>
</tbody>
</table>

Table 2.24 shows only the publications with the largest number of papers related to the topic being studied. The first three positions are occupied by journals: Information and Software Technology (4 papers), IEEE Transactions on Software Engineering (3 papers), and Empirical Software Engineering (2 papers) which together represent almost 25% of the total. The conferences with the highest number of papers are the International Conference on Program Comprehension, the International Symposium on Empirical Software Engineering and International Symposium on Empirical Software Engineering and Measurement, all of them with 2 papers, and each one of them representing nearly 6% of the total. The workshop with the highest number of papers is the International Workshop on Visualizing Software for Understanding and Analysis, with 2 papers.

Table 2.25. Diagrams obtained from RE.

<table>
<thead>
<tr>
<th>Diagrams from RE</th>
<th>Number of studies</th>
<th>Percentage</th>
<th>List of papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>10</td>
<td>15.15%</td>
<td>[P25], [P26], [P27], [P28], [P30], [P36], [P37]</td>
</tr>
<tr>
<td>No</td>
<td>56</td>
<td>84.85%</td>
<td>[P1], [P2], [P3], [P4], [P5], [P6], [P7], [P8], [P9], [P10], [P11], [P12], [P13], [P14], [P15], [P16], [P17], [P18], [P19], [P20], [P21], [P22], [P23], [P24], [P29], [P31], [P32], [P33], [P34], [P35], [P48]</td>
</tr>
<tr>
<td>Total</td>
<td>66</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is also important to note that only 15.15% of the systems used in the empirical studies are obtained from a reverse engineering (RE) process, while 84.85% of the diagrams used are created during the development process (Table 2.25).
2.6. DISCUSSION

This systematic mapping study has discovered 38 relevant papers (containing 66 empirical studies) in peer-reviewed journals, conferences, and workshops, and has classified them in order to obtain responses to the research questions presented, which are briefly summarized below:

- **RQ1 asked:** Which diagrams are most frequently used in studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams? The results show a clear order, which indicates the relative importance that researchers attach to 3 diagram types when they study the maintenance of the UML diagrams themselves or also how they study the use of UML diagrams when performing maintenance on the source code: class diagrams (34%), sequence diagrams (17%) and statechart diagrams (16%). Some studies performed partial comparisons of the understandability of one type of diagram versus another. The three aforementioned diagrams are reported to contribute most to understandability. The low occurrence of studies relating to use case diagrams (8%) could be explained by the fact that there are no studies addressing this type of diagrams which are directly related to maintenance tasks (they are always presented with other UML diagrams). This low rate could also be related to the origin of the diagrams. In some cases (about 15%), the diagrams are obtained from the code by using reverse engineering. In this case, the use case diagrams are not generally available. Furthermore, owing to their high level of abstraction, use cases say nothing about the structure of the system, and hence do not contain information that a maintainer needs to perform changes/modifications which tend to be a lower level of information as regards detail.

- **RQ2 asked:** Which variables are investigated in the empirical studies? How are they measured? Most of the empirical studies found which focused on the maintenance of the UML diagrams themselves concentrated on measuring the understandability of class diagrams (23%) or statechart diagrams (13%). The measures used for this dependent variable are related to the time spent by the subjects in understanding the UML diagrams and the subjects’ effectiveness when performing the understandability tasks. There are some more isolated studies focusing on the use of the UML diagrams in maintenance tasks. In these cases, the measures used are, apart from time, the correctness of the solutions proposed and the quality of the code. It is supposed that a better understanding of the diagram correlates with a better understanding of the system, and that this should positively influence the maintenance of the source code. However, sufficient work with which to validate this assumption is not available. More studies are needed which deal with the influence of UML diagrams on the maintainability of source code.

- **RQ3 asked:** What is the state-of-the-art in empirical studies concerning the maintenance of UML diagrams or the maintenance of source code when using UML diagrams? To answer RQ3, an analysis based on different perspectives of the empirical literature in the field is presented. The analysis is presented from the following three perspectives:
**How?:**

- **How is the maintenance of the UML diagrams studied?**
  
  Most of the studies that were found present results of controlled experiments (95%). This is a well-known way in which to validate data, but the field would benefit (in terms of generalizability) from the additional performance of case studies. Industrial data or real projects should be analyzed to confirm the results obtained in the laboratory context.

**Where?:**

- **Where are the empirical studies carried out?**
  
  We now know that most of the studies performed are controlled experiments. These studies can be considered as only academic results, since they were carried out in a laboratory context (83%), so it is also necessary to perform more empirical studies in industrial contexts to corroborate the academic results.

**What?:**

- **What types of subjects have been used in empirical studies?**
  
  The subjects that performed the tests are mostly students (78%). A minority of studies involved members of the university teaching staff (11%) or practitioners (12%). This fact reveals that more empirical studies with practitioners are necessary to strengthen the external validity of the results. It would thus be feasible to ascertain whether or not the findings obtained with students also hold for practitioners.

- **What is maintained in the empirical studies?**
  
  If we focus on the results obtained in this systematic mapping study, we can see that most of the studies are related to the maintenance of only the UML diagrams themselves (83%), rather than to the UML diagrams and the code (14%). It is also important to highlight that most of the diagrams used represent prototypes of systems or very simple systems (74%). Using diagrams from true complex systems when performing maintenance tasks would help to test whether the UML has specific benefits. It is also important to note that those experiments in which tasks are related to the maintenance of the code, rather than simply maintaining a diagram, are more representative of the current situation in industry. There should be more studies which deal with the maintenance of the UML diagrams themselves as part of maintaining an entire system. There also needs to be a comparison of the maintainers’ performance when using up to date diagrams as opposed to using older versions of the documentation - e.g., that originate from the design of a system. In our opinion, the degree of correspondence between diagrams and code could have an influence...
on some maintenance tasks. It would be logical to expect to obtain better results when this correspondence is high.

- **What are the treatments in the empirical studies?**
  Most of the empirical studies (42%) attempt to compare different aspects of the UML diagrams, for example diagrams with stereotypes vs. without them, different levels of details in the diagrams, etc.

- **What is the quality of the papers found?**
  More than half of the papers (53%) obtained a relatively high score in this quality assessment (note that all of the papers related to the maintenance of source code when using the UML are in this category), and only 9% obtained a low quality score. The fact that most of the papers obtain the maximum points related to the description of the aims of the research and its context is worth noting. None of the primary studies, in contrast, provide a reference to the raw data used to test the results.

- **RQ4 asked: Which of the factors studied influence the maintainability of a system (source code and diagrams)?** These results are summarized in a classification tree of factors in Figure 2.2. It will be observed that the presence of some specific diagrams, such as the use of stereotypes and a good, correct layout, positively influences the maintainability of a system. There are also other factors, such as a high structural complexity of the system, a low level of detail in diagrams, a high nesting level of composite states or the presence of defects in diagrams, which have a negative influence on the maintainability of a system.

  It is also noteworthy that only two papers are specifically related to empirical studies concerning the use of the UML in maintenance (modification) tasks:

  - The first of these is [P5], which presents the results of two controlled experiments carried out with students from different universities. [P5] reports that the time taken to make changes in the source code is less when the UML diagrams are used than when they are not used, while if the time taken to perform the corresponding modifications to diagrams is included, there is no significant difference. In both cases, however, the quality of the modifications is greater when the subjects have UML diagrams.

  - [P8] presents the results of a controlled experiment carried out with professionals. In both this paper and that mentioned above, the time taken to perform the modifications to the system, the time spent on maintaining the diagrams and the quality of the proposed modifications are measured. This study [P8] does not find any significant difference in the time spent on performing changes, but the authors do find that the quality of the changes is higher for the group of subjects with UML diagrams, as is the case in [P5].
2.7. THREATS TO VALIDITY

We have classified the threats to validity on this study by following the classification provided by Wohlin et al. (Wohlin et al., 2012). The main threats to the validity of a systematic mapping study are publication selection bias (construct validity), inaccuracy in data extraction (construct validity), and misclassification (conclusion validity) (Sjøberg et al., 2005).

With regard to the *construct validity*, we considered six digital sources, which included journals, conferences and workshops which are relevant to software engineering. The scope of journals and conferences covered in this systematic mapping study is sufficiently wide to attain a reasonable completeness in the field studied. We did not include additional papers such as grey literature (technical reports, PhD thesis, etc.), and limited ourselves to peer-review publications. We believe that we have achieved a reasonably complete coverage, as most grey literature either has its origins in peer-reviewed papers or appears in what will eventually become peer-reviewed papers; it may, however, be the case that both of these circumstances are true for a given piece of grey literature. Some relevant papers might exist which have not been included (which it might be possible to extract with the use of a snowballing process), although our knowledge of this subject is such that we do not believe that there are many of these. We performed an automated search on 6 digital libraries in order not to rule out papers from conferences or journals which deal with topics of interest but may not be well-known sources. This could be a threat to the validity of this work because manual searchers seem to be more helpful than those which are automated but this requires a previous knowledge of the source used in the search, as is presented in the results of (Kitchenham et al., 2010), but this work was published after we had performed the search, and we have not therefore been able to take these results into consideration. To help ensure an unbiased selection process, we defined research questions in advance, organized the selection of papers as a multistage activity, involved three researchers in this activity and documented the reasons for inclusion/exclusion, as suggested in (Liu et al., 2005). As was discussed above, the decisions to select the papers to be included as primary studies in this systematic mapping study were made by multiple researchers, and rigorous rules were followed. A further challenge was that there is no keyword standard that we are aware of which distinguishes between different quality characteristics, nor are there methods in empirical software engineering that could be used to extract quality characteristics and research methods in a consistent manner.

Moreover, the duplication of papers is a potential threat to frequency counts and to the statistics in this systematic mapping study. The structure of the database managed by the SLR-Tool (Fernández-Sáez et al., 2010), which was used to perform this systematic mapping study, is designed to handle duplication, but one threat would be that of duplication going undetected. However, at least two individuals have read through all the relevant papers without detecting further duplicates. We also found it quite difficult to manage the duplication of empirical studies performed by the same author but which are reported as a part of other studies, i.e., different papers had a part
of their contents in common. We examined them exhaustively in order to attempt to
detect whether or not they were the same study, following a fixed procedure, but the
elimination or otherwise of possible duplications might be a threat.

The fact that we also considered the term understandability as an alternative term
for maintainability, which a priori is not a real synonym or sub-characteristic of it
based on the ISO 25000 (ISO/IEC, 2008), might be a threat of the validity of our
work. However, we based our decision on the results of previous works which judge
understandability to be a factor that influences maintainability (Briand et al., 2001a;
Deligiannis et al., 2002; Genero et al., 2007; Harrison et al., 2000).

With regard to conclusion validity, we would like to comment that when extracting
data from papers there is a certain degree of subjectivity in terms of what is and what
is not determined to be related. Furthermore, bias can affect the interpretation of the
results. The data was extracted from the papers by one researcher and checked by
another. When necessary, disagreements were resolved through discussion by
involving the third author. Data extraction and classification from prose is difficult at
any time, and the lack of standard terminology and standards could very well result in
a misclassification. We believe, however, that the extraction and selection activity was
rigorous and that it followed the guidelines provided in (Brereton et al., 2007). The
use of multiple experts to perform the classification also reduced the risk of
misclassification.

2.8. CONCLUSIONS

More than fifteen years on from when the UML was first introduced in 1997, it
would be useful for the software industry to gather empirical evidence of use of the
UML in the software development life cycle, specifically in software maintenance,
which is the most resource-consuming phase. With that need to gather such
information in mind, this paper presents a systematic mapping study on empirical
studies performed as regards the use of UML diagrams in the maintenance of source
code and also on the maintenance of only the UML diagrams themselves. This
systematic mapping study covers papers published in journals, conferences and
workshops, found via six digital libraries in the period between January 1997 and
March 2010.

The systematic manner in which this systematic mapping study was carried out, by
following the guidelines provided in (Kitchenham and Charters, 2007), makes this
study rigorous and fair.

We would like to highlight two problems that were dealt with during the process of
the systematic mapping study:

- It is not usually possible to judge the relevance of a study from a review of the
  abstract alone. The standard of IT and software engineering abstracts is too poor to
  rely on when selecting primary studies, and this makes it necessary to review the
  full text. When used properly, structured abstracts are very useful in improving the
  quality and usefulness of the abstract (Budgen et al., 2011b). Structured abstracts
  must contain the following sections: 1) Context (the importance and relevance of
the research), 2) Objectives (the main objectives pursued), 3) Methods (the research method followed and the proposal provided to attain the objectives), and 4) Results (the main findings and conclusions obtained).

- The search engines have some limitations when performing the search on the abstract alone, or when the search string is quite complex, and could not therefore be searched directly. The search string thus had to be tailored to each digital library by splitting the original and combining the results manually. Current search engines are not designed to support systematic literature reviews. Unlike medical researchers, software engineering researchers need to perform resource-dependent searches.

During this systematic mapping study we attempted to answer one main research question: *What is the current existing empirical evidence with regard to the use of UML diagrams in source code maintenance and the maintenance of the UML diagrams themselves?*

We found only two papers ([P5] and [P8]) which were able to help us to answer the first part of this question. Two controlled experiments in [P5] report how the presence of UML diagrams can help to reduce the time needed to maintain the source code. These two experiments and the experiment presented in [P8] show that the quality of the modifications made by subjects is greater when UML diagrams are available. Although the existing studies related to the use of UML diagrams in source code maintenance are in favour of using the UML for this kind of tasks since the quality of the modifications is greater when these diagrams are available, few papers concerning this issue have been published.

If we focus on the papers which deal with the maintenance of only the UML diagrams themselves, we detected some studies which present empirical results concerning the benefits of using UML diagrams as opposed to simply using text, or how the availability of some specific diagrams (class, sequence, state, activity and use case diagrams) can be a positive factor in the maintenance of source code. We also found several pieces of research concerning the maintenance of the UML diagrams themselves that reported some factors which can improve that maintenance of these diagrams (such as the use of stereotypes, the use of composite states, the use of a correct level of detail or of a correct layout), and which will eventually influence the maintenance of the software system. We also found studies concerning how factors that are external to the system under maintenance might influence its maintenance, such as the maintainers’ experience and ability.

The main findings according to the categories used to classify the 38 selected primary studies are:

- Research method: Most of the studies present the results of controlled experiments.
- Context: Most of the experiments are carried out in a laboratory context.
- Subjects: Most of the experiments are performed by Computer Science undergraduates.
Dependent variable: The most common dependent variable used in the empirical studies is the maintainability of class diagrams which is usually measured using time and accuracy.

Available diagrams: The most widely-used diagrams in the studies selected are class and sequence diagrams.

Object to maintain: Most of the studies focus on maintaining only the diagrams.

Type of system: Synthetic systems are those most often used in the studies found.

Origin of diagrams: Most of the studies found use diagrams that are not obtained from a reverse engineering process.

Treatments: Most of the empirical studies compare different aspects of UML diagrams, for example diagrams with stereotypes vs. those without them, different levels of detail in the diagrams, etc.

Quality of papers: Almost 90% of the studies have a medium or high quality.

Almost all the studies with regard to the study of the maintenance of the UML diagrams themselves found are experiments that compare different aspects of UML diagrams, but their external validity, i.e., their generalizability, is questionable given the material, tasks and subjects used.

In summary, one of the main findings is that there is a need for studies that take into account the measurement of cost and productivity, which are variables that have great repercussions in industrial contexts. In order to strengthen the external validity, i.e., the generalizability of the empirical results, we suggest that more experiments and case studies should be carried out in industrial contexts, with real systems and maintenance tasks performed by practitioners under real conditions. Studies concerning how to improve the understandability of a UML diagram (and hence the maintainability of the source code) are carried out from different points of view, comparing different variables. In addition, the maintenance of both diagrams and diagrams and code together must be considered in future empirical studies. It is important to note the lack of empirical studies under real conditions, owing to the fact that the majority of the studies presented used toy systems or prototypes. Due to those reasons, we wish to stress the need for further empirical studies carried out in industrial contexts to investigate whether the use of the UML can lead to important differences that make the costs involved worthwhile, particularly as regards source code maintenance.

We suggest that the Software Engineering community should share or exchange available resources, i.e., models and code, using existing repositories (for example, ReMoDD (France et al., 2006)). After collecting the documentation of some systems and selecting the most representative ones, a benchmark could be created in order to make the results of future empirical studies directly comparable. A repository with experimental material would also help researchers to provide more empirical results by generating new studies or replicating the existing ones.

While conducting this systematic mapping study we detected some studies which present empirical results concerning the benefits of using UML diagrams (activity, class, sequence, statechart and use case diagrams) as opposed to simply using text, or
on how the availability of some specific diagrams can be a positive factor in the maintenance of source code. We also discovered several pieces of research concerning the maintenance of the UML diagrams themselves which report some factors that can improve the maintenance of the system.

We would also like to provide references to some of the papers which obtained high scores in the quality assessment (about 30 out of 40 points) and which could be used as examples of good experiments: P8, P10, P14, and P20.

To conclude this paper, we trust that the systematic mapping study published herein will serve both as a guide to past research in the area, and as a foundation for future research. This work is also an attempt to support other researchers and practitioners by providing a library of papers on empirical evidence concerning the use of UML diagrams in the maintenance of both source code and the UML diagrams themselves.

Acknowledgements: This research has been funded by the following projects: MEDUSAS (CDTI-MICINN and FEDER IDI-20090557), ORIGIN (CDTI-MICINN and FEDER IDI-2010043(1-5)), PEGASO/MAGO (MICINN and FEDER, TIN2009-13718-C02-01), and GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01).
CHAPTER 3. DOES THE LEVEL OF DETAIL (LOD) OF UML DIAGRAMS INFLUENCE SOFTWARE MAINTENANCE?

ABSTRACT

Although the UML is considered to be the de facto standard notation with which to model software, there is still resistance to model-based development. UML modeling is perceived to be expensive and not necessarily cost-effective. It is therefore important to collect empirical evidence concerning the conditions under which the use of UML makes a practical difference. The focus of this paper is to investigate whether and how the Level of Detail (LoD) of UML diagrams impacts on the performance of maintenance tasks in a model-centric approach. A family of experiments consisting of one controlled experiment and three replications has therefore been carried out with 81 students with different abilities and levels of experience from 3 countries (The Netherlands, Spain, and Italy). The analysis of the results of the experiments indicates that there is no strong statistical evidence as to the influence of different LoDs. The analysis suggests a slight tendency toward better results when using low LoD UML diagrams, especially if used for the modification of the source code, while a high LoD would appear to be helpful in understanding the system. The participants in our study also favored low LoD diagrams because they were perceived as easier to read. Although the participants expressed a preference for low LoD diagrams, no statistically significant conclusions can be drawn from the set of experiments. One important finding attained from this family of experiments was that the participants minimized or avoided the use of UML diagrams, regardless of their LoD. This effect was probably the result of using small software systems from well-known domains as experimental materials.

Keywords: UML diagrams, software maintenance, level of detail, controlled experiment, replication, family of experiments.

3.1. INTRODUCTION

The maintenance phase is the phase of the life cycle that absorbs a significant number of software development resources (Glass, 2002; Pressman, 2005): “Maintenance typically consumes 40 percent to 80 percent of software costs. Therefore, it is probably the most important life cycle phase of software” and “60 percent of the budget is spent on software maintenance, and 60 percent of this maintenance is to enhance existing software”. It is therefore important to attempt to improve maintainers’ performance by helping them to understand the requirements and the system being maintained (especially in the case of critical systems). The comprehension of a program could consume half of the time spent by developers on maintenance (Fjeldstad and Hamlen, 1979). The presence of proper documentation might thus help them in this part of the maintenance (Tryggeseth, 1997). Program comprehension may be particularly benefited by graphical documentation (Arisholm and Briand, 2006; Dzidek et al., 2008) since, as is commonly stated, “a picture says more than 1000 words”. Several modeling languages have therefore emerged, which are either domain-specific modeling languages or general-purpose modeling languages. They appeared as a means to improve the customer’s understanding and, in general, communication among team members (Nugroho and Chaudron, 2009). The
wide application of modeling has led to the development of numerous informal and formal approaches for modeling, such as Entity Relationship Diagrams (ERD) (Wieringa, 2003) with which to model data, Specification and Description Language (SDL) (Glässer et al., 2003) with which to model telecommunication systems, or formal modeling languages such as Z (Spivey, 1989) and B (Abrial, 1996). The Unified Modeling Language, or UML (OMG, 2010), is an Object Oriented modeling notation which first appeared in 1997 and has now become one of the most widely used modeling languages in industry and the de-facto software modeling notation (Erickson and Siau, 2007; Grossman et al., 2005). However, there is still frequent resistance to model-based development in many software organizations since the use of the UML is perceived to be expensive and not necessarily cost-effective (Arisholm et al., 2006). This is even worse for organizations that use agile methods to develop software (Cohen et al., 2004). It is thus important to investigate whether the use of the UML can make a practical difference and justify the costs sustained. It is also important to study under which conditions (for example, in which type of software projects, with which programming languages, type of maintainers, type of maintenance, size of system, and so on) the UML can make a practical difference.

Obviously, not all UML diagrams have the same complexity, layout, level of abstraction, etc. Furthermore, previous studies have shown that the style and rigor used in the diagrams may vary considerably according to the software project (Lange et al., 2006) and affect the source code of the system in a different way (Nugroho and Chaudron, 2008). On the one hand, the different purposes for which a diagram may be intended (for example: architecting solutions, communicating design decisions, detailed specification for implementation, or automatically generating implementation code) signifies that the same system can be represented with different styles. On the other hand, forward design diagrams, i.e., the diagrams generated during forward development, are sometimes available for maintainers in the maintenance phase, but when this is not the case, the diagrams may be reconstructed using a Reverse Engineering (RE) technique. RE diagrams are easy to obtain without having to invest in a lot of developer effort, as opposed to forward design diagrams which need to be manually generated and updated, and the level of detail of the RE diagrams is extremely high. Given the ease of the generation of RE diagrams and the fact that they can be generated automatically at any time, it is possible for maintainers to have up-to-date diagrams modeling the system when they need them. However, their Level of Detail might be a factor to be considered in order to ensure the effectiveness of the diagrams during software maintenance. If is not possible to reverse engineer a UML diagram, then it is necessary to create it by hand during the development phase (to avoid the need to create it during later maintenance). A question therefore arises: “To what LoD is it necessary to detail the diagrams as part of the document of a system process being used in order not to use more resources in its creation than the paybacks that might originate from it in a future maintenance based on a model-centric approach?” Our experience allows us to state that the use of incomplete documentation and of a subset of the entire software system on which a maintenance operation impacts is quite common in the software industry. A similar question to the
previous one then arises: “To what LoD is it necessary to update the UML documentation in order for it to be synchronized with the source code and thus attain understandability benefits during further maintenance?

The aforementioned considerations motivated us to perform a family of experiments in order to analyze whether and how the understandability and modifiability of source code vary when using low or high LoD UML diagrams during the maintenance of a system when a model-centric approach is used. The UML diagrams were only used to support the understanding of the system and to guide its implementation but not to automatically generate the source code. We focused on a model-centric approach rather than an MDD (Model-Driven Development) approach because the LoD expected to be used with the purpose of automatically generating source code would probably always be sufficiently high for details in the source code not to be lost, but this is not the hypothesis of this paper.

We considered a specific type of maintenance in order to restrict the context of this family of experiments. We specifically selected perfective maintenance for two reasons: (i) it appears to be one of the most commonly used types of maintenance (systems are constantly evolving to become more complete and to offer more functionalities); and (ii) the requirements of this type of maintenance could be easily reproduced when using systems from well-known domains.

The family consisted of a controlled experiment and three replications carried out with students from 3 different countries (The Netherlands, Spain, and Italy). The participants were 81 subjects with different abilities and levels of experience with the UML and with the Java programming language (the experimental material, is available at: http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/).

The main goal of this paper is to present a thorough description of this family of experiments, the main findings and the practical implications.

This paper is organized as follows. The related work is presented in Section 3.2. Section 3.3 presents the family of the experiments, while the results obtained from the experiment and the three replications are discussed in Section 3.4. The meta-analysis and the summary and discussion of the data analysis are presented in Sections 3.5 and 3.6, respectively. The threats to validity and the lessons learnt are highlighted in Section 3.7. The paper concludes in Section 3.8 with final remarks and future work.

3.2. RELATED WORK

In (Fernández-Sáez et al., 2013a) a systematic mapping study of empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code is presented. This systematic mapping study discovered 38 papers (published from January 1997 to January 2010) containing 66 empirical studies (primary studies) related to this topic. Only the following two works are directly related to the use of UML diagrams in source code maintenance:

- In the work of Dzidek (Dzidek et al., 2008) an experiment was performed to investigate whether the use of the UML influences maintenance in comparison to
the use of source code only. This experiment investigated the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real nontrivial system, using 20 professional developers as subjects. These developers had to perform 5 maintenance tasks consisting of adding new functionalities to the existing system, and the correctness, time and quality of the solution were measured. Both source code and UML diagrams, when available, had to be maintained. The results of this work show a positive influence of the presence of the UML for maintainers. In terms of time, the UML subjects took longer if the UML documentation had to be updated, but that difference was not statistically significant. However, the UML was always beneficial in terms of functional correctness (introducing fewer faults into the software), because the subjects in the UML group obtained, on average, a practically and statistically significant 54 percent increase in the functional correctness of changes. The UML also helped produce better quality code when the developers were not yet familiar with the system. This experiment is a replication of a previous work performed with students which is presented by Arisholm (Arisholm et al., 2006) and obtains similar results. In our experiments we compare the influence of two different kinds of UML diagrams on source code maintainability rather than the presence of UML diagrams per se. Moreover, the UML diagrams in our experiments do not need be updated.

- Arisholm et al. (Arisholm et al., 2006) presented the results of a controlled experiment carried out to assess the impact of UML design diagrams on software maintenance. Software professionals were involved. The authors analyzed the time taken to perform the modifications to the system, the time spent maintaining the models, and the quality of the modifications performed. The results of the quantitative analysis revealed no significant difference in the time spent making the modifications. Similarly to Dzidek’s work (Dzidek et al., 2008), they observed that the quality of the modifications was higher for those participants who had been given UML diagrams. As in Dzidek’s work (Dzidek et al., 2008), the participants’ ability and experience were not analyzed as regards the comprehensibility and modifiability of source code. One difference with regard to our study is that the authors analyzed the effect of UML based documentation (a use case diagram, sequence diagrams for each use case, and a class diagram) on modification tasks performed both on UML diagrams and source code.

The remaining primary studies in the systematic mapping study were focused on the understandability of UML diagrams. This topic is, in some respects, related to the understandability of the system since one influences the other, but these kinds of studies are not considered in this paper.

We updated the search period to August 2013 and found several additional empirical studies that were to some extent related to the content of the present paper, and these are described as follows:

- Scanniello et al. (Scanniello et al., 2014) used a family of controlled experiments to discover that the use of UML analysis diagrams (those obtained in an early phase
or the development process, such as the requirements elicitation or analysis phase) does not significantly improve the comprehension and modifiability of source code with regard to the use of source code alone. Analysis diagrams might be considered as low LoD diagrams while design diagrams might be considered as high LoD diagrams. The most remarkable difference between that paper and the work presented herein is that here we consider the UML diagrams produced in the design phase with different levels of detail.

- In the work of Karahasanovic (Karahasanovic and Thomas, 2007) the experiment performed is focused on the comprehension and the difficulties involved in maintaining object-oriented systems. During this experiment the subjects, who were 34 students in their third year of a computer science degree, had to perform 3 different maintenance tasks on a medium-size object-oriented system written in Java. The tasks were related to extending, updating and deleting the functionalities of the system, i.e., maintaining the system. It is important to note that the subjects had to give higher priority to the quality of the solutions than to a shorter development time, which may have influenced the results of the experiment. UML diagrams were also presented to the subjects of the experiment, and the correctness of their solutions was measured, but this work was only focused on exploring the participants’ strategies and problems while they were conducting maintenance tasks on an object-oriented application. Two major groups of difficulties were related to the comprehension of the application structure, namely the understanding of GUI implementation and OO comprehension and programming.

- With regard to the influence of the LoD of UML diagrams in software development, Nugroho presents an empirical experiment focused on the understandability of UML diagrams with different LoDs (Nugroho, 2009). The paper in question measures the correctness and efficiency of 53 computer science Master’s degree students as regards comprehending UML diagrams. The results show a better understanding of diagrams when they have a high LoD. There are some differences between the work presented by Nugroho and the family of experiments summarized in this paper. On the one hand, the experiment presented by Nugroho focuses solely on the comprehension of UML diagrams and the subjects did not have the source code of the system. On the other hand, Nugroho focuses on their use in the development phase and not on maintenance.

- In a preliminary study by Scanniello (Scanniello et al., 2012) the results of an experiment to assess whether the comprehension of source code is affected when it is added to UML class and sequence diagrams produced in the design phase is presented. The results reveal that the participants benefited from the use of UML diagrams. An average improvement of 14% was achieved when the participants accomplished the comprehension task with the class and sequence diagrams, but the time needed to comprehend the code was not significantly influenced from a statistical point of view. Our work differs from that of Scanniello (Scanniello et al., 2012) because we compare low LoD diagrams with high LoD diagrams rather than with no UML diagrams, and the dependent variable is also different because Scanniello’s work (Scanniello et al., 2012) focuses on the comprehension of the
source code and the work presented here is focused on the maintainability of source code. Bigger systems are also used here.

- A comparison of the attitude and performance of maintainers when using Forward Engineered (FD) diagrams vs. Reverse Engineered (RE) diagrams during the maintenance of source code is shown by Fernández-Sáez (Fernández-Sáez et al., 2013b) through a family of controlled experiment with students (40, 51 and 78 respectively). The statistical results, show a tendency to obtain better results when using UML diagrams (concretely class diagrams), that were hand-made during the design phase. When considering the qualitative results of the post-experiment survey, it was also noteworthy that the subjects preferred FD diagrams when understanding and maintaining a system. The post-experiment survey results also led the authors to conclude that the subjects found RE diagrams, and particularly the sequence diagrams, difficult to understand.

All the related work is summarized in Table 3.1, thus providing the reader with an overview of the main information in order to compare the empirical studies. The columns in the table are described as follows:

- **Ref:** contains the reference to the paper that presents the empirical study considered.
- **Type of empirical study:** indicates the type of empirical study summarized in the paper (a survey, an experiment, a family of experiments, etc.)
- **Goal:** describes the goal pursued by the empirical study.
- **Subjects:** presents the numbers of subjects who participated in the empirical studies and the type of subjects (students, professionals, academic staff, etc.).
- **Independent variables:** describes the variables that are studied to ascertain their effect on the dependent variables. The values (treatments) of the independent variables are also presented.
- **Dependent variables:** presents the outcome variables, which are the variables that are affected by the changes produced in the independent variables.
- **Experiment design:** contains the type of design selected, which can be between-subjects (each subject receives only one treatment) or within subjects (each subject receives all the treatments).
- **Tasks:** describes the tasks to be performed by the subjects as part of the empirical study.
- **Results:** reveals the main findings obtained.

The analysis of the literature presented above reveals that, with regard to the usefulness of UML diagrams in helping during source code maintenance, empirical evidence is scarce (see summary in Table 3.1). Nonetheless, the existing evidence shows that there is, to some extent, a favorable tendency in that the use of UML diagrams benefits source code maintenance (Arisholm et al., 2006; Dzidek et al., 2008).
Table 3.1. Summary of related works (part 1/2).

<table>
<thead>
<tr>
<th>Ref</th>
<th>(Dzidek et al., 2008)</th>
<th>(Arisholm et al., 2006)</th>
<th>(Scanniello et al., 2014)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of study</td>
<td>1 experiment</td>
<td>2 experiments</td>
<td>Family of experiments (1+3)</td>
</tr>
<tr>
<td>Goal</td>
<td>To investigate whether the use of UML influences maintenance in comparison to the use of only source code.</td>
<td>To investigate whether the use of UML influences maintenance in comparison to the use of only source code.</td>
<td>To investigate whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code.</td>
</tr>
<tr>
<td>Subj</td>
<td>20 professional developers.</td>
<td>Undergraduate students (22 and 76, respectively).</td>
<td>Undergraduate students (24, 22, 22 and 18, respectively).</td>
</tr>
<tr>
<td>Independent variables</td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The use of UML analysis models (possible values: presence or absence of UML diagrams accompanying source code).</td>
</tr>
<tr>
<td>Dependent variables</td>
<td>-Time needed to change source code. -Time needed to change source code + UML diagrams. -Functional correctness and quality of the solution.</td>
<td>-Time needed to change source code. -Time needed to change source code + UML diagrams. -Correctness of the change. -Quality of the change.</td>
<td>-Comprehension level of the source code. -Capability of a maintainer to modify source code.</td>
</tr>
<tr>
<td>Experiment design</td>
<td>Between-subject design.</td>
<td>Between-subject design.</td>
<td>Within participants counterbalanced experimental design.</td>
</tr>
<tr>
<td>Tasks</td>
<td>Modification tasks in source code and in UML diagrams.</td>
<td>Modification tasks in source code and in UML diagrams.</td>
<td>Comprehension and modification tasks and subjective questions.</td>
</tr>
<tr>
<td>Results</td>
<td>The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system.</td>
<td>The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system.</td>
<td>UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.</td>
</tr>
</tbody>
</table>
### Table 3.1. Summary of related works (part 2/2).

<table>
<thead>
<tr>
<th>Ref</th>
<th>(Nugroho, 2009)</th>
<th>(Karahasanovic and Thomas, 2007)</th>
<th>(Scanniello et al., 2012)</th>
<th>(Fernández-Sáez et al., 2013b)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of study</strong></td>
<td>1 experiment</td>
<td>1 experiment</td>
<td>1 experiment</td>
<td>Family of experiments (1+2)</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>To determine the influence of different levels of detail (LoD) of UML diagrams in source code maintenance.</td>
<td>To investigate the comprehension and the difficulties involved in maintaining object-oriented systems.</td>
<td>To investigate whether the comprehension of source code increases when participants are provided with UML class and sequence diagrams produced in the software design phase</td>
<td>To determine the influence of the origin of UML diagrams in source code maintenance.</td>
</tr>
<tr>
<td><strong>Subj.</strong></td>
<td>11 undergraduate students.</td>
<td>34 under-graduate students.</td>
<td>16 undergraduate students.</td>
<td>Undergraduate students (40, 51 and 78 respectively)</td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td>The level of detail of UML diagrams (possible values: high or low LoD).</td>
<td>-</td>
<td>The use of sequence and class diagrams created in the design phase (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The origin of UML diagrams (possible values: forward design or reverse engineered diagrams).</td>
</tr>
<tr>
<td><strong>Dependent variables</strong></td>
<td>Understandability and modifiability of source code.</td>
<td>Comprehension and Modification of system.</td>
<td>Comprehension of source code.</td>
<td>Understand-ability and modifiability of source code.</td>
</tr>
<tr>
<td><strong>Experiment design</strong></td>
<td>Between-subjects balanced design.</td>
<td>Between-subject design.</td>
<td>Within-subjects.</td>
<td>Between-subjects balanced design.</td>
</tr>
<tr>
<td><strong>Tasks</strong></td>
<td>Comprehension and modification tasks and subjective questions.</td>
<td>Modification questions.</td>
<td>Comprehension questions.</td>
<td>Modification tasks + subjective questions.</td>
</tr>
<tr>
<td><strong>Results</strong></td>
<td>No significant results in favor of high or low LoD. There is a slight tendency in favor of low LoD diagrams.</td>
<td>Detection of common difficulties as regards understanding when maintaining software systems.</td>
<td>Participants comprehend source code significantly better when class and sequence diagrams are added together.</td>
<td>Tendency to obtain better results when using class diagrams, that were hand-made during the design phase based on statistical and subjective results. Sequence reversed engineered diagrams were considered difficult to read.</td>
</tr>
</tbody>
</table>
The completeness of the documentation, which is significantly related to the level of detail of a UML diagram, has been identified as one of the most important quality factors to affect the overall quality of the documentation (Garousi et al., 2013). The influence of the different levels of detail of UML diagrams during software development has been studied (Nugroho, 2009), but there are no clear results as to their influence on software maintenance because analysis models do not influence maintenance (Scanniello et al., 2014) while design models do (Scanniello et al., 2012).

What is more, the design diagrams obtained good results when compared to reverse engineered diagrams (Fernández-Sáez et al., 2013b). In our case, we are of the opinion that low LoD diagrams are similar to analysis models, while high LoD diagrams are comparable to design models. The issues that are taken into account to state this similarity are explained in section 3.3.1. Given that analysis diagrams are more similar to low LoD diagrams and that design diagrams are more similar to high LoD diagrams, better results are expected when using high LoD diagrams.

The research presented in this paper is different from previous research because it pursues a different goal. In particular, we aim to discover whether high LoD diagrams help maintainers to better understand the system and perform the modifications that need to be made to the source code during the maintenance phase in comparison with low LoD diagrams. We shall thus contribute, to some extent, towards building up the body of knowledge regarding the evidence of the benefits of using UML.

### 3.3. THE FAMILY OF EXPERIMENTS

Families of experiments allow researchers to answer questions that are beyond the scope of individual experiments and permit the generalization of the findings from various studies, thus providing evidence with which to confirm or reject specific hypotheses (Basili et al., 1999). Replications of empirical studies might be regarded as an essential activity in the construction of knowledge in any empirical science based on the following propositions: “We do not take even our own observations quite seriously, or accept them as scientific observations, until we have repeated and tested them” (Pressman, 2005) and “… [replication] is needed not merely to validate one’s findings, but more importantly, to establish the increasing range of radically different conditions under which the findings hold, and the predictable exceptions” (Lindsay and Ehrenberg, 1993).

We carried out a family of experiments to investigate whether the different levels of detail (LoD) affect the work that must be carried out by a maintainer. The main goal of this paper is to present a family of experiments that consists of one experiment and three external replications. Figure 3.1 provides some information (i.e., the name of the experiment, the number of participants, and the context) about the experiments and their chronology.

In this family of experiments the participants were undergraduate/graduate students from different years of the Bachelor’s or Master’s Computer Science degree. The original experiment (denominated as E-UL) was carried out at the University of Leiden (The Netherlands) in March 2010. The first replication (denominated as R1-
UCLM) was performed at the University of Castilla-La Mancha (Spain) in April 2010. Two more replications (denominated as R2-UB and R3-UB, respectively) were carried out in May 2010 at the University of Bari (Italy).

![Figure 3.1. Chronology of the family of experiments.](image)

The original experiment (E-UL) was performed as a pilot study with a small number of subjects, in order to obtain an initial insight into the influence of the LoD (Fernández-Sáez et al., 2012). We decided to corroborate the results obtained by carrying out a series of external replications. These experiments can be considered as strict replications of the original experiment because the essential aspects of the experimental conditions have not varied in any way: the only difference is the participants involved.

In order to run and report this family of experiments, we followed the recommendations and guidelines for conducting and reporting empirical research in software engineering provided in several works (Carver, 2010; Jedlitschka et al., 2008; Juristo and Moreno, 2001; Wohlin et al., 2000). For replication purposes, we created an experimental package containing the experimental material, which is available at:

http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/

The main characteristics of each experiment and the replications are described in the following subsections, including goal, context selection, variables selection, hypothesis formulation, experimental design, experimental tasks, experimental procedure, analysis procedure and documentation and communication.

### 3.3.1. Goal

The main goal of the family of experiments, using the GQM template (Basili and Weiss, 1984; Basili et al., 1999), is to: “Analyze the level of detail in UML diagrams with the purpose of evaluating it with regard to the understandability and modifiability of source code during a model-centric development from the point of view of researchers, in the context of Computer Science undergraduate/graduate students at the following Universities: University of Leiden, University of Castilla-La Mancha and University of Bari”.
As in (Nugroho, 2009), we considered that the LoD in UML diagrams should be defined as the amount of information that is used to represent a modeling element. LoD is a 'continuous' metric, but in the experiment we have taken two “extremes” - high and low LoD. Based on the LoD concept, one of our main assumptions is that the more information a diagram contains, the more is known about the concepts/knowledge described in the diagram. This knowledge would help maintainers to better understand the source code of the system and consequently modify it according to the maintenance requirements.

We decided to use 3 different types of diagrams (use case, sequence and class diagrams) since they are those most frequently used (Dobing and Parsons, 2006; Erickson and Siau, 2007; Grossman et al., 2005). When the LoD used in a UML diagram is low, it typically contains only a few syntactical features, such as class-name and associations, without specifying any further facts about the class. When it is high, the diagram also includes class attributes and operations, association names, association directionality, and multiplicity. In sequence diagrams, in which there is a low LoD, the messages among objects have an informal label, and when the LoD is high the label is a method name plus the parameter list. We do not consider that it is possible to distinguish between low and high LoD in use case diagrams because they are very simple diagrams, so both groups received the same set of diagrams. The elements that fit each level of detail are detailed in Table 3.2, and an example of the two versions of a class diagrams is shown in the APPENDIX D.

<table>
<thead>
<tr>
<th>Diagram</th>
<th>Element</th>
<th>Low LoD</th>
<th>High LoD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class diagram</td>
<td>Classes (box and name)</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Attributes</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Types in attributes</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Operations</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Parameters in operations</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Associations</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Association directionality</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Association multiplicities</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Aggregations</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Compositions</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Sequence diagram</td>
<td>Actors</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Objects</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Messages in informal language</td>
<td>✔️</td>
<td>☒️</td>
</tr>
<tr>
<td></td>
<td>Messages with formal language (name of a method)</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Parameters in messages</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td></td>
<td>Labels in return messages</td>
<td>☒️</td>
<td>✔️</td>
</tr>
<tr>
<td>Use case diagrams</td>
<td>They were the same for both treatments because they do not contain enough details to distinguish between Low or High LoD</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
3.3.2. Context selection

The experimental material used in E-UL was in English. However, the material was translated for the replications. A native Spanish speaker translated all the material into Spanish in the case of R1-UCLM, while a native Italian speaker translated all the material into Italian in the case of R2-UB and R3-UB. The replicators supported the native speakers and helped them when needed (e.g., in the translation of technical terms).

Two experimental objects were used:

- **System A**: a library application from which a user can borrow books.
- **System B**: a sport center application from which users can rent services (tennis courts, etc.)

System A is a library extracted from (Ericksson et al., 2004). We decided to use this because it was a representative system, it was complete (source code and diagrams were available) and it gave us a starting point with which to compare our results. It was only possible to compare the results obtained from the subjects who received System A with a high LoD with the result from (Karahasanovic and Thomas, 2007) because the material received was the same (although the tasks were different). The other treatments, i.e., the documents related to System A with a high LoD, were created by us for the experiment). The source code in System A contains 3369 lines of code and the documentation consists of 5 use case diagrams with 17 use cases, 5 class diagrams with 25 classes and 15 sequence diagrams with 242 messages (Table 3.3). System B is a Sport center application created as part of the Master’s degree Thesis of a student from the University of Castilla-La Mancha. The source code in System B contains 5123 lines of code and the documentation consists of 5 use case diagrams with 22 use cases, 4 class diagrams with 16 classes and 21 sequence diagrams with 226 messages (Table 3.3).

<table>
<thead>
<tr>
<th></th>
<th>#Class diagrams</th>
<th>#classes</th>
<th>#Sequence diagrams</th>
<th>#messages</th>
<th>#Use case diagrams</th>
<th>#use cases</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>System A</td>
<td>5</td>
<td>25</td>
<td>15</td>
<td>242</td>
<td>5</td>
<td>17</td>
<td>3369</td>
</tr>
<tr>
<td>System B</td>
<td>4</td>
<td>16</td>
<td>21</td>
<td>226</td>
<td>5</td>
<td>22</td>
<td>5123</td>
</tr>
</tbody>
</table>

Both systems are desktop applications from well-known domains and are more or less the same size, based on the metrics mentioned below. Owing to their origins and size we consider them as small systems, based on the size classification given in (Scanniello et al., 2010). The documentation of both systems can be considered to be sufficiently realistic for small-sized development projects of the following kinds: in-house software (the system is developed inside the software company for its own use) or sub-contracted (a sub-contractor develops or delivers part of a system to a main contractor) (Lauesen, 2002). They can also be considered as realistic systems because they were written in Java, which is one of the most common programming languages
used in companies (based in our experience). We considered well-known domains in order to avoid the extra cognitive effort required in the case of non-well-known domains, which might have biased the results.

As explained previously, the documentation of System A was extracted from a book about UML (Ericksson et al., 2004) from which the RUP (Rational Unified Process) process is taught, and it was therefore expected that the diagrams would be created by following that modeling process. System B, meanwhile, was created as part of a Master’s degree Thesis by following an RUP process. In both cases, therefore, the diagrams and the source code were developed by adopting an incremental development process similar to that suggested by Bruegge and Dutoit (Bruegge and Dutoit, 2010). In both cases the diagrams were created with the intention of supporting the understanding of the system and guiding the implementation of the source code within the aforementioned model-centric development processes but the intention was not related to the automatic generation of source code from it. As in the majority of developments of this kind, the diagrams were available in order to provide insights into how the system had been created and to increase the understandability in possible further maintenance. UML diagrams may therefore provide a "lens" for the code that may assist with maintenance tasks, but it is important to highlight that the UML diagrams used during a model-centric development would not faithfully represent the reality of the source code.

In order to check the material used in the experiment, one of the authors first reviewed the documentation of the two systems to find possible issues. No modifications of any note were needed to improve either the documentation (e.g., typographical errors from the models were removed) or the source code (e.g., the source code was indented). Next, and with the intention of checking the complexity and duration of the experiment, a pilot study was carried out with 3 PhD students from the University of Leiden and 2 PhD students from the University of Castilla-La Mancha. Owing to the fact that the intended participants in the family of experiments would have less experience than those who took part in in the pilot study (Master’s Degree students and PhD students respectively), we decided to provide more time in the execution of the experiments than in the pilot. Based on the results of these two pilots, we considered that the two experimental objects fitted the time constraints of the experiment while being sufficiently realistic as regards the small maintenance operations that novice software maintainers perform within a software company (Cohen et al., 2004).

The experimental material delivered to the subjects consisted of the UML diagrams (use case, class and sequence diagrams) and the JAVA code of Systems A and B, the paper-based answer sheets and the post-experiment questionnaire.

We conducted all the experiments in research laboratories under controlled conditions. The subjects’ knowledge was sufficient for them to understand the systems provided, and they had roughly the same background. The participants, who were grouped by experiment, had the following characteristics:
Chapter 3: Does the Level of Detail (LoD) of UML Diagrams Influence Software Maintenance?

- **E-UL**: The experiment was carried out by 11 Computer Science students from the University of Leiden (The Netherlands) who were taking the *Software Engineering* course in the second-year of their B.Sc. Almost all the students were non-repeating course students (only one was repeating the course) and virtually none of them had any industrial experience (one had experience as a programmer and another as a designer). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. Based on their responses to the preliminary test, which contained background questions, we assumed that most of the subjects considered their own UML knowledge to be medium for use case and class diagrams, and low for sequence diagrams. Moreover, most of the subjects’ knowledge of JAVA language was medium/high, while all their knowledge of C++ language was medium/high, and this language is very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.

- **R1-UCLM**: The first replication was carried out by 16 Computer Science students from the University of Castilla-La Mancha (Spain) who were taking the *Quality of Information Systems* course in the second year of their Master’s degree. More than 80% of the students were non-repeating course students and virtually none of them had any industrial experience (three had experience as programmers and another as a maintainer). Their knowledge of UML diagrams had been acquired during other courses (such as Software Engineering I and Software Engineering II). The subjects’ responses in the preliminary test again showed that most of them considered their own UML knowledge to be medium/high for use case, class and sequence diagrams. All of them had medium/high knowledge of JAVA language, and they also had knowledge of C and C++ languages which are very similar languages to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.

- **R2-UB**: The second replication was carried out by 32 Computer Science students from the University of Bari (Italy) who were taking the *Software Engineering* course in the second year of their B.Sc. Three quarters of the students were non-repeating course students and virtually none of them had any industrial experience (two has experience as programmers). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. The responses provided in the preliminary test led us to assume that most of the subjects considered their own UML knowledge to be medium/high for class diagrams, and medium/low for use case and sequence diagrams. Almost half of them considered their knowledge of JAVA language to be medium/high, while the rest considered it to be low. In addition, almost all of them considered their knowledge of C and C++ languages to be medium/high, and these languages are very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out.
R3-UB: The third replication was carried out by 22 Computer Science students from the University of Bari (Italy) who were taking the Software Engineering course in the second year of their B.Sc. 80% of the students were non-repeating course students and virtually none of them had any industrial experience (three had experience as programmers and another as a maintainer). Their knowledge of UML diagrams was general and had been acquired during the Software Engineering course. Most of the subjects considered their own UML knowledge to be medium/high for use case and class diagrams, and medium/low for sequence diagrams (based on the results of the preliminary test). Most of them considered their knowledge of JAVA language to be medium/high, while half of them considered their knowledge of C and C++ languages to be medium/high, and these languages are very similar to JAVA. Their knowledge of UML and JAVA was reinforced in a training session on UML diagrams and JAVA organized to take place the day before the experiment was carried out. The students who participated in the whole family of experiments were volunteers who were selected for convenience (the students available in the corresponding course). The subjects who participated in the experiment were not graded on their performance, and they obtained extra points in their final marks.

The participants selected were students in their last academic year because they can be considered as novice software engineers in real companies. They were required to work as individual maintainers.

3.3.3. Selection of variables

The independent variable (also called the “main factor”) is the LoD, which is a nominal variable with two values: low LoD and high LoD.

The dependent variables are:

- **Modifiability**, denoting the capability of the source code to be modified by a software engineer (modeler, designer, maintainer, etc.).
- **Understandability**, denoting the capability of the source code to be comprehended by a software engineer (modeler, designer, maintainer, etc.).

These two variables were selected because understandability and modifiability are considered in literature to have a direct influence on maintainability (Arisholm et al., 2006; Cruz-Lemus et al., 2010; ISO/IEC, 2014; Roehm et al., 2012). The following measures were defined in order to measure these dependent variables:

- **Understandability Effectiveness** ($UEffec$): This measure reflects the ability to correctly understand the system presented. It is calculated with the following formula: $\frac{\text{number of correct answers}}{\text{number of questions}}$.
- **Modifiability Effectiveness** ($UEffic$): This measure reflects the ability to correctly modify the system presented. It is calculated with the following formula: $\frac{\text{number of correctly performed modification tasks}}{\text{number of modification tasks}}$. 
• **Understandability Efficiency (MEffec):** This measure also reflects the ability to correctly understand the system presented. It is calculated with the following formula: \( \text{number of correctly answered questions} / \text{time spent} \).

• **Modifiability Efficiency (MEffic):** This measure also reflects the ability to correctly modify the system presented. It is calculated with the following formula: \( \text{number of correctly performed tasks} / \text{time spent} \).

These measures were applied to the responses obtained from a questionnaire composed of 3 understandability questions and 3 modifiability questions (further details of this are provided in Section 3.3.3.6.) in order to obtain a quantitative evaluation of these dependent variables. A higher value of the measures reflects a better understandability/modifiability.

Additional independent variables (called “co-factors”) were considered according to the experimental design selected, and their effect was controlled and analyzed:

• **Order.** We analyzed whether the order in which the LoD were used by the subjects biased the results. This was analyzed because of the design selected (see Table 3.4), i.e., the variation in the order of application of each method (low LoD, high LoD), which was used with the intention of alleviating learning effects.

• **System.** This factor indicates the systems (i.e., A and B) used as experimental objects. The design selected for the experiment (see Table 3.4) forced us to choose two application domains in order to avoid learning effects. Although our intention was that the systems used would not be a confounding factor owing to their size similarities and domains, this might not have been that case, and the system might have influenced the subjects’ performances. We therefore studied the effect of the systems used on the results obtained. The following diagrams were used during the experiment:
  o A-H: high LoD diagrams of system A.
  o A-L: low LoD diagrams of system A.
  o B-H: high LoD diagrams of system B.
  o B-L: low LoD diagrams of system B.

• **Ability.** A quantitative assessment of the participants’ ability was obtained by computing the average of the grades of the courses taken. The students from the University of Leiden and Castilla La Mancha with average grades below 8/10 (a 5 is needed to pass the course) were classified as low ability participants, otherwise high. In the replications conducted with students at the University of Bari, the threshold was 27/30. Students with average grades below 27/30 (an 18 is needed to pass the course) were therefore classified as low, otherwise high. We used different threshold values in the experiments conducted in The Netherlands, Spain and Italy because different grading systems are used in these countries. We considered this cofactor in our efforts to investigate whether subjects’ abilities play any role in the maintenance of source code, i.e., we discriminated between users according to their respective levels of Ability with the purpose of testing the hypothesis that this is a relevant influencing factor that should be taken into account when adopting these kinds of diagrams.
3.3.4. Hypotheses formulation

Based on the assumption explained above, that the more information a diagram contains, the more is known about the concepts/knowledge described in the diagram, the following 4 null hypotheses have been formulated and tested:

- **H_1_0**: There is no significant difference in the subjects’ understandability effectiveness when working with UML diagrams modeled using high or low levels of detail.

- **H_2_0**: There is no significant difference in the subjects’ understandability efficiency when working with UML diagrams modeled using high or low levels of detail.

- **H_3_0**: There is no significant difference in the subjects’ modifiability effectiveness when working with UML diagrams modeled using high or low levels of detail.

- **H_4_0**: There is no significant difference in the subjects’ modifiability efficiency when working with UML diagrams modeled using high or low levels of detail.

The goal of the statistical analysis will be to reject these null hypotheses and possibly to accept the alternative ones (e.g., H_n1=¬ H_n0). All the hypotheses are two sided because we did not postulate that any effect would occur as a result of the LoD.

3.3.5. Experimental design

When designing the experiment we attempted to alleviate several issues that might threaten the validity of the research being carried out by considering the suggestions provided in (Wohlin et al., 2000). We selected a balanced factorial design in which the group-interaction acted as a confounding factor (Kirk, 1995). This ensured that each subject worked on different experimental objects (System A or B) in two runs, using a different LoD (High or Low) each time. This design permits learning and fatigue effects to be mitigated and allows the effect of cofactors to be studied. Table 3.4 presents the outline of the experimental design.

<table>
<thead>
<tr>
<th>RUN 1</th>
<th>LoD</th>
<th>RUN 2</th>
<th>LoD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td><strong>Group 1</strong></td>
<td><strong>Group 2</strong></td>
<td><strong>Group 3</strong></td>
</tr>
<tr>
<td>A</td>
<td>Group 1</td>
<td>Group 2</td>
<td>A</td>
</tr>
<tr>
<td>B</td>
<td>Group 3</td>
<td>Group 4</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 3.4. Experimental design.

All the experiments are balanced with regard to the number of participants assigned to the method (LoD). The assignment of the subjects to each group was performed by using Ability as a balancing factor, i.e., before carrying out the experiment or the replications we provided the subjects with a background questionnaire and assigned them to the 4 groups, randomly distributing experience equally throughout the groups (blocked design by experience) in an attempt to alleviate experience effects.
3.3.6. Experimental tasks

We asked the subjects to perform the following three kinds of tasks (all of them written on paper and not using computers):

- **Understandability tasks**: This block of tasks consisted of answering 3 multiple choice questions concerning the semantics of the system, i.e., the semantics of diagrams and the semantics of code. These questions were multiple choice questions and were used to obtain $UEffec$ and $UEffic$. An example of this kind of question is shown in Figure 3.2. Each of these tasks had a score of 1 point when correct and 0 when false.

![Figure 3.2. Example of Understandability task.](image)

- **Modifiability tasks**: The intention of these tasks, which are also called perfective maintenance tasks, was to extend the functionality of the system and to improve the services provided (Lientz and Swanson, 1980). In our case, new functionalities had to be added to the system, with the subjects receiving a list of 3 independent new requirements (Table 3.5) which had to be addressed by modifying the system code and thus adding/changing certain functionalities.

Neither of the systems used had special maintenance needs because they were not used in production, and the maintenance tasks were therefore created specifically for this family of experiments. We attempted to create a list of requirements by adding functionalities that seemed to be logical requirements (owing to the fact that they had come from well-known domains). We choose these types of maintenance tasks because they are frequent in real environments and they are also doable by individual maintainers in the time with which the participants were provided. Other types of maintenance, like corrective maintenance (i.e., activities intended to remove errors or bugs from the software), might be more frequent but could require a lot of time to detect where it is necessary to work in the case of isolated maintainers. These tasks were used to calculate the modifiability measures ($MEffec$ and $MEffic$). The results of these modifiability tasks have different complexities: from adding/changing a couple of lines of code to adding the source code of a complete class. The modifiability tasks needed to be answered by using data collection forms, i.e., templates or answer sheets which had to be filled in with pieces of code. We used these data collection forms to obtain a structured response which facilitated the correction of the results. The subjects were provided with
answer sheets to allow them to structure their responses related to the maintenance tasks. The reason for doing this was that maintaining source code on paper is not easy owing to space constraints, so the subjects were required to write changes to the source code in a structured manner on the answer sheets (format: line-number, change type, Java code, etc.). They had to fill in a different form depending on the element that they wished to maintain (a class, an attribute, etc.). An example of one of the tasks is shown on Figure 3.3.

Table 3.5. Summary of modifiability tasks.

<table>
<thead>
<tr>
<th>System</th>
<th>Task</th>
<th>Summary of modifiability task descriptions</th>
<th>Maximum mark (in points)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>T1</td>
<td>The library system should store its borrowers’ emails.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>A ticket showing a borrower’s loans at a specific time should be generated by the system.</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>T3</td>
<td>The information about requests to buy new titles should be stored by the system.</td>
<td>6</td>
</tr>
<tr>
<td>B</td>
<td>T1</td>
<td>The sport center system should store its customers’ telephone numbers.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>T2</td>
<td>A ticket showing a customer’s reservations at a specific time should be generated by the system.</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>T3</td>
<td>The information about the sport center’s instructors should be stored by the system.</td>
<td>6</td>
</tr>
</tbody>
</table>

Figure 3.3. Example of Modifiability tasks for System B.

The other tasks, along with all the material used in the experiment, are available at: http://alarcos.esi.uclm.es/maintenanceUMLfamilyExperimentsLoD/

The greatest change consisted of adding a class which would need the declaration of at least 22 new statements. In general, between 1 and 3 classes needed to be modified. The complexity of the tasks might not appear to be too complex owing to the number of needed statements which have to be changed, but the complexity of the task lies in the difficulty involved in detecting where the change to the source code should be made, along with how it should be carried out. It should also be
borne in mind that 6 tasks (3 understandability and 3 modifiability tasks) had to be completed in 2 hours, using systems that the subjects had never seen before. We limited the time of the experiment to fit in with the subjects’ time availability. They were only required to maintain the source code, i.e., they did not need to update diagrams according to their changes or to create test cases.

Table 3.6. Post experiment questions.

<table>
<thead>
<tr>
<th>Id</th>
<th>Question/Issue</th>
<th>Possible Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>The difficulty of the tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q2</td>
<td>The training was sufficient to be able to perform the tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q3</td>
<td>The clarity of the material provided</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q4</td>
<td>The task objectives were perfectly clear to me</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q5</td>
<td>The tasks I performed were perfectly clear to me</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q6</td>
<td>I did not experience difficulty in reading the diagrams</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q7</td>
<td>I did not experience difficulty in reading the source code</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q8</td>
<td>The use of high level of detail UML diagrams helps in maintenance tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q8.1</td>
<td>The use of attributes in class diagrams help you to perform maintenance tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q8.2</td>
<td>The use of operations in class diagrams help you to perform maintenance tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q8.3</td>
<td>The use of formal method names in messages in sequence diagrams help you to perform maintenance tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q8.4</td>
<td>The use of parameters in messages in sequence diagrams help you to perform maintenance tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Q9</td>
<td>How much time (as a percentage) did you spend looking at the diagrams in order to perform the software maintenance?</td>
<td>Multiple choice question</td>
</tr>
<tr>
<td>Q10</td>
<td>How much time (as a percentage) did you spend source code browsing?</td>
<td>Multiple choice question</td>
</tr>
<tr>
<td>Q11</td>
<td>The time for performing the experiment was adequate</td>
<td>Multiple choice question</td>
</tr>
</tbody>
</table>

1 = strongly agree; 2 = agree; 3 neutral; 4 = disagree; 5 = strongly disagree (Q2, Q4 - Q8.4)

1 = very difficult; 2 = difficult; 3 = medium; 4 = easy; 5 = very easy (Q1)

1 = very clear; 2 = clear; 3 = correct; 4 = unclear; 5 = very unclear (Q3)

A = more time needed; B = less time needed; C = enough time (Q11)

A. <20%; B. >=20% and <40%; C. >=40% and <60%; D. >=60% and <80%; E. >=80% (Q9, Q10)
• **Post-experiment tasks:** At the end of the execution of each run, the subjects were additionally asked to fill in a post-experiment questionnaire consisting of 15 questions (see Table 3.6) whose goal was to obtain feedback about the subjects’ perception of the experiment execution, which could be used to explain the results obtained. The answers to the questions were based on a five-point Likert scale (Oppenheim, 2000) or multiple choice questions.

### 3.3.7. Experimental procedure

In order to check the material and the duration of the experiment, a pilot study was carried out with 3 PhD students from the University of Leiden before carrying out the actual experiment (Fernández-Sáez et al., 2012). The results of the pilot study were used as a basis to adapt the number of tasks and their complexity to the experimental time constraints. Some spelling mistakes were also corrected, and some requirement statements were rewritten in order to make them more understandable. A pilot experiment took place before each replica in order to check that the material had been correctly translated. The Spanish version of the material was checked by 2 PhD students at the University of Castilla-La Mancha, and the Italian version was checked by one member of the University of Bari’s academic staff.

The experiment and replications took place in two sessions of two hours each:

- **Training session:** The subjects first attended a training session in which detailed instructions on the experiment were presented and the main concepts of UML and JAVA were revised. No details of the experimental hypotheses were provided in this session, and the subjects carried out an exercise similar to those in the experimental tasks in collaboration with the instructor. During the training session, the subjects were required to fill in a background questionnaire. The participants were informed that the data collected in the experiments would be used for research purposes and treated confidentially, and that their grade on the course they were taking would not be affected by the grade obtained in the experiment. After the training session, we assigned the participants to one of the 4 groups in accordance with the marks obtained in the background questionnaire, thus obtaining balanced and homogeneous groups (see Table 3.4).

- **Execution session:** The execution of the experiment took place in the second session, in a classroom, in which the students were supervised by the course instructor (a different one depending on the replication) and one experimenter (always the same one). This second session consisted of two runs, as is required by the selected experimental design shown in Table 3.4. In each run, each of the groups was given a different treatment, i.e., they first received the material for the first run, and when they had finished, they received the first post-experiment questionnaire. After filling in the post-experiment questionnaire, the material for the second run was provided. When the second run had finished, the subjects received the last post-experiment questionnaire. The participants were not allowed to interact during either each laboratory run or while passing from the first run to the second. We did not provide details on the experimental hypotheses, and
informed the participants that their grade on the course would not be affected by their performance.

After the execution of each experiment or replication, the data collected were placed on an excel sheet, following an answering diagram constructed before the experiment was carried out. On this sheet, the answers to the understanding tasks were graded as correct or incorrect, since they were multiple choice questions. With regard to the modifiability tasks, each task has a maximum mark (see Table 3.5), depending on the correctness of the answer provided. This means that for each task, a mark was given to the subject depending on the number of correct lines of code added to the solution. Incorrect answers were not graded negatively, i.e., lines of code which do not solve the tasks.

3.3.8. Analysis procedure

The data analysis was carried out by considering the following steps:

1. Analysis of the main factor:
   1.1. We first carried out a descriptive study of the measures of the dependent variables, i.e., understandability and modifiability (see Section 3.4.1).
   1.2. We then analyzed the characteristics of the data to determine which parametric or non-parametric test would be most appropriate. We performed a Kolmogorov-Smirnov test (Sheskin, 2007) to determine the normality of distributions and a Levene test (Sheskin, 2007) to determine the homogeneity of variances.
   1.3. Based on the results of the previous test, we then tested the null hypotheses formulated using the non-parametric Wilcoxon test (Conover, 1998) for the data collected in the experiment (see Sections 3.4.2.1 to 3.4.2.4). The use of this test was possible because, in accordance with the design of the controlled experiment, we obtained paired samples.
   1.4. To strengthen the results of each experiment, we decided to integrate them using a meta-analysis (see Section 3.4.2.5). A meta-analysis is a set of statistical techniques with which to combine the different effect sizes of the experiments in order to obtain a global effect of a factor on a dependent variable.

2. Analysis of the cofactors:
   2.1. **System**: The influence and the interaction of the System cofactor with the LoD (main factor) is analyzed in Section 3.4.3. There could be several concerns as regards performing this kind of analysis when adopting the within-participants counterbalanced design (Kitchenham et al., 2002). We therefore used interaction plots (Devore and Farnum, 1999) to study the interaction of LoD with the cofactors. Interaction plots are simple line graphs on which the means of the dependent variables (in our case $UEffec$, $UEffic$, $MEffec$ and $MEffic$, one on each graph) for each level of a factor (in our case the cofactor System) are plotted over all the levels of another factor (in our case the LoD).
The resulting lines are parallel when there is no interaction and nonparallel when an interaction is present.

2.2. **Order**: The measurement differences of the dependent variables when participants used high or low LoD were analyzed by using the method suggested by Briand et al. (Briand et al., 2005). This analysis is called Order of method (one of the cofactors), and the results are shown in Section 3.4.4. In particular, let:

- \( \text{Diff}(\text{low}) \) be the differences in the values for a dependent variable that the participants achieved when performing the tasks using a low LoD first and a high LoD second, and

- \( \text{Diff}(\text{high}) \) be the differences in the values for a dependent variable that the participants achieved when performing the tasks using a high LoD first and a low LoD second.

In order to verify whether \( \text{Diff}(\text{high}) \) was statistically greater than \( \text{Diff}(\text{low}) \), we used the non-parametric Mann–Whitney test (Conover, 1998) for independent samples. In our case, we expected that \( \text{Diff} \) (high) would be greater than \( \text{Diff} \) (low) for all the measures \( \text{UEffec}, \text{UEffic}, \text{MEffec} \) and \( \text{MEffic} \). The rationale was that the participants might obtain higher scores when using a high LoD in the first run, and we therefore tested the null hypothesis \( H_{0\text{Order}}: \text{Diff}(\text{low}) = \text{Diff}(\text{high}) \). In the case of rejecting this hypothesis, we verified whether \( \text{Diff}(\text{high}) \) was greater than \( \text{Diff}(\text{low}) \) for all the measures (i.e., \( \text{Diff}(\text{high}) > \text{Diff}(\text{low}) \)).

2.3. **Ability**: The influence of the Ability cofactor was analyzed by performing non-parametric tests owing to the nature of the data obtained (see Section 3.4.5).

3. The data collected from the post-experiment questionnaire with the participants’ subjective perceptions was eventually analyzed using descriptive statistics and illustrated with column graphs (see Section 3.4.6).

In all the statistical tests performed, we decided (as is customary) to accept a 5% probability of committing a Type-I-Error (Wohlin et al., 2000) and used SPSS (SPSS, 2003) as a statistical package.

### 3.3.9. Documentation and communication

Issues such as documentation (Juristo et al., 2013) and communication among experimenters (Vegas et al., 2006) may influence the success or the failure of replications. These issues were handled by using laboratory packages and knowledge sharing mechanisms. The material was originally written in English, and was translated into Spanish and Italian for the corresponding replications. The material included: the background questionnaire, the understanding/modification questionnaires, the answer sheets, the source code and the UML diagrams (two versions: high and low LoD) and the post-experiment questionnaire. The groups of
Chapter 3: Does the Level of Detail (LoD) of UML Diagrams Influence Software Maintenance?

The experimenters also shared a document to provide a common background in order to communicate all the terms related to the design and analysis of the experiment.

The experimenters began with an initial face-to-face meeting at which the principal ideas of the experiments were discussed and reported in minutes. All the experimenters then exchanged the minutes of the meeting by e-mail in order to agree to a shared common research plan. This phase was relevant to sharing knowledge among the experimenters and to discussing possible issues related to the study.

The experimenters used instant messaging tools and e-mails to establish a communication channel in all the phases of the study. Teleconferences also took place in order to share knowledge among the research groups and to discuss the experimental procedure that the participants had to follow.

3.4. RESULTS

In this section, we present the data analysis following the procedure presented above: the presentation of the descriptive statistics, the test of the hypotheses related to the main factor (LoD), the analysis of the influence of cofactors and the analysis of the post-experiment questionnaire.

3.4.1. Descriptive statistics and exploratory analysis

Table 3.7, Table 3.8 In the case of R2-UB1, the subjects obtained, on average, better results in both measures related to understandability ($UEffec$ and $UEffic$) when using high LoD UML diagrams and, on the contrary, they obtained better results in both measures related to modifiability ($MEffec$ and $MEffic$) when using low LoD diagrams.

Table 3.9, and Table 3.10 show the respective descriptive statistics of the $UEffec$, $UEffic$, $MEffec$, and $MEffic$ measures (i.e., number of subjects (N), mean ($\bar{X}$), standard error (SE), and standard deviation (SD)), grouped by LoD.

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Low LoD</th>
<th>High LoD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>$\bar{X}$</td>
</tr>
<tr>
<td>E-UL</td>
<td>11</td>
<td>0.76</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>16</td>
<td>0.58</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>32</td>
<td>0.54</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>22</td>
<td>0.38</td>
</tr>
</tbody>
</table>

As will be observed in Table 3.7, Table 3.8, In the case of R2-UB1, the subjects obtained, on average, better results in both measures related to understandability ($UEffec$ and $UEffic$) when using high LoD UML diagrams and, on the contrary, they obtained better results in both measures related to modifiability ($MEffec$ and $MEffic$) when using low LoD diagrams.

Table 3.9, and Table 3.10 (first row of each table), in half of the cases in E-UL ($MEffec$ and $MEffic$), the subjects obtained, on average, better results when using low
LoD UML diagrams. Only in the case of *UEffic* are the results better with high LoD diagrams. In the case of *UEffec*, there were no differences in the means. This group had the same tendency as the subjects of R1-UCLM who, on average, obtained better results in all the measures when using low LoD UML diagrams.

**Table 3.8. Descriptive statistics for UEffic.**

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Low LoD</th>
<th></th>
<th></th>
<th>High LoD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>( \bar{x} )</td>
<td>SE</td>
<td>SD</td>
<td>N</td>
<td>( \bar{x} )</td>
</tr>
<tr>
<td>E-UL</td>
<td>11</td>
<td>0.0030</td>
<td>0.00046</td>
<td>0.0016</td>
<td>11</td>
<td><strong>0.0037</strong></td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>16</td>
<td><strong>0.0025</strong></td>
<td>0.00032</td>
<td>0.0013</td>
<td>16</td>
<td>0.0023</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>32</td>
<td>0.0018</td>
<td>0.00030</td>
<td>0.0017</td>
<td>31 (1 outlier)</td>
<td><strong>0.0020</strong></td>
</tr>
<tr>
<td>R3-UB2</td>
<td>22</td>
<td><strong>0.0016</strong></td>
<td>0.00041</td>
<td>0.0019</td>
<td>22</td>
<td>0.0016</td>
</tr>
</tbody>
</table>

In the case of R2-UB1, the subjects obtained, on average, better results in both measures related to understandability (*UEffec* and *UEffic*) when using high LoD UML diagrams and, on the contrary, they obtained better results in both measures related to modifiability (*MEffec* and *MEffic*) when using low LoD diagrams.

**Table 3.9. Descriptive statistics for MEffec.**

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Low LoD</th>
<th></th>
<th></th>
<th>High LoD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>( \bar{x} )</td>
<td>SE</td>
<td>SD</td>
<td>N</td>
<td>( \bar{x} )</td>
</tr>
<tr>
<td>E-UL</td>
<td>11</td>
<td><strong>0.44</strong></td>
<td>0.066</td>
<td>0.22</td>
<td>11</td>
<td>0.40</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>16</td>
<td><strong>0.61</strong></td>
<td>0.047</td>
<td>0.19</td>
<td>16</td>
<td>0.58</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>31</td>
<td><strong>0.47</strong></td>
<td>0.050</td>
<td>0.28</td>
<td>31</td>
<td>0.43</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>22</td>
<td><strong>0.52</strong></td>
<td>0.049</td>
<td>0.23</td>
<td>22</td>
<td>0.47</td>
</tr>
</tbody>
</table>

**Table 3.10. Descriptive statistics for MEffic.**

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Low LoD</th>
<th></th>
<th></th>
<th>High LoD</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>N</td>
<td>( \bar{x} )</td>
<td>SE</td>
<td>SD</td>
<td>N</td>
<td>( \bar{x} )</td>
</tr>
<tr>
<td>E-UL</td>
<td>11</td>
<td><strong>0.0049</strong></td>
<td>0.00052</td>
<td>0.0017</td>
<td>11</td>
<td>0.0045</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>16</td>
<td><strong>0.0056</strong></td>
<td>0.00052</td>
<td>0.0021</td>
<td>16</td>
<td>0.0051</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>30</td>
<td><strong>0.0039</strong></td>
<td>0.00055</td>
<td>0.0030</td>
<td>30</td>
<td>0.0038</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>22</td>
<td>0.0033</td>
<td>0.00038</td>
<td>0.0018</td>
<td>22</td>
<td><strong>0.0037</strong></td>
</tr>
</tbody>
</table>

A mixture of results also appeared in the case of R3-UB2, in which the subjects obtained, on average, better results when using low LoD UML diagrams in the *UEffic* and *MEffec* measures. In the other two variables, *UEffec* and *MEffic*, the results were better with high LoD diagrams.

These results show the following:

- *UEffec*: on average, the results show a tendency in favor of high LoD in two of the experiments and another in favor of low LoD (in the other there are no differences in favor of either low or high LoD).
- *UEffic*: The results are similar to those of the *Ueffec*, i.e., an average tendency in favor of high LoD in two of the experiments and another two in favor of low LoD.
• MEffec: on average, the participants achieved slightly better results when employing low LoD diagrams because better mean values for low LoD were achieved in all the experiments. The difference in favor of low LoD is more evident for R2-UB1 and R3-UB2.

• MEffec: on average, the participants achieved slightly better results when employing low LoD diagrams. Better mean values for low LoD were achieved in all the experiments, with the exception of R3-UB2.

In summary, it can be observed that when the subjects used low LoD diagrams they obtained better values in both of the measures related to modifiability (MEffec and MEffic). This indicates that low LoD diagrams may, to some extent, improve the modification of the source code. We cannot reach any conclusion related to the understandability because the differences between both groups (high and low LoD groups) are less evident.

3.4.2. Influence of LoD

In order to test the formulated hypotheses (H1,0, H2,0, H3,0, H4,0) we analyzed the effect of the main factor (i.e., LoD) on the measures of the dependent variables considered (i.e., UEffec, UEffic, MEffec and MEffic) using the Wilcoxon test. We performed this test owing to the characteristics of the data, i.e., in most cases the data were not normal and there was no homogeneity of variances (conclusions extracted after a Kolmogorov-Smirnov test and a Levene test, respectively).

The results for each measure of the Wilcoxon test are shown in the tables (Table 3.11, Table 3.12, Table 3.13, and Table 3.14) in the following subsections, in which the #obs column describes the number of observations, the influence column refers to the existence of the influence of the LoD (the p-value for the statistics test is between brackets). The statistical observed power and the size of the effect are then shown. These tables also report the number of participants that achieved better results using a low LoD (# of low>high) or a high LoD (# of low<high). The number of participants that obtained the same results using a low or high LoD are also shown (# of low=high).

### 3.4.2.1. Testing Understandability Effectiveness (H1,0)

The results in Table 3.11 suggest that the null hypothesis (H1,0) cannot be rejected when using the Wilcoxon test since the p-value is greater than 0.05 in the first experiment and in all the replications. This means that there is no significant difference in UEffec in either group (low or high LoD diagram groups).

<table>
<thead>
<tr>
<th>Dep. Variable</th>
<th>#obs</th>
<th>Influence (low&gt;high)</th>
<th>Statistical Power</th>
<th>Effect size</th>
<th># of low&gt;high</th>
<th># of low=high</th>
<th># of low&lt;high</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>22</td>
<td>No (0.680)</td>
<td>0.051</td>
<td>0.001</td>
<td>2/11 (18%)</td>
<td>6/11 (55%)</td>
<td>3/11 (27%)</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>32</td>
<td>No (0.500)</td>
<td>0.058</td>
<td>0.003</td>
<td>5/16 (31%)</td>
<td>7/16 (44%)</td>
<td>4/16 (25%)</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>64</td>
<td>No (0.661)</td>
<td>0.076</td>
<td>0.003</td>
<td>9/32 (28%)</td>
<td>11/32 (34%)</td>
<td>12/32 (38%)</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>44</td>
<td>No (0.130)</td>
<td>0.271</td>
<td>0.043</td>
<td>6/22 (27%)</td>
<td>5/22 (23%)</td>
<td>11/22 (50%)</td>
</tr>
</tbody>
</table>
We decided to investigate this result in greater depth by calculating the number of subjects who achieved better values when using low or high LoD models (i.e., a low LoD value is higher than a high LoD value or vice versa, respectively). As Table 3.11 shows, the number of subjects who obtained the same results for both treatments (high and low LoD) is relatively high in E-UL and R1-UCLM. There were more subjects who performed better with a high LoD than with a low LoD (with the exception of R1-UCLM), but the differences in comparison to the opposite group are very small in the case of E-UL (only one subject). The results suggest that there was no statistical significant difference in the \textit{UEffic} when the participants employed low or high LoD models. Nonetheless, the statistical powers are very low, probably because of a small effect size, so we would be assuming a 0.729 to 0.949 (i.e., 1 – statistical power) estimated probability of a Type II error in our assertions. Given the low value of the observed power we cannot obtain strong conclusions.

3.4.2.2. Testing Understandability Efficiency (H2,0)

As is shown in Table 3.12, the p-value is greater than 0.05 in all the cases, so the null hypothesis (H2,0) cannot be rejected. This means that there is no significant difference in the \textit{UEffic} in either group (low or high LoD group).

<table>
<thead>
<tr>
<th>Dep. Variable</th>
<th>#obs</th>
<th>Influence (low≠high)</th>
<th>Statistical Power</th>
<th>Effect size</th>
<th># of low&gt;high</th>
<th># of low=high</th>
<th># of low&lt;high</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>22</td>
<td>No (0.594)</td>
<td>0.180</td>
<td>0.056</td>
<td>6/11 (55%)</td>
<td>0/11 (0%)</td>
<td>5/11 (45%)</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>32</td>
<td>No (0.776)</td>
<td>0.077</td>
<td>0.008</td>
<td>9/16 (56%)</td>
<td>1/16 (6%)</td>
<td>6/16 (38%)</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>64</td>
<td>No (0.104)</td>
<td>0.081</td>
<td>0.005</td>
<td>12/32 (38%)</td>
<td>1/32 (3%)</td>
<td>19/32 (59%)</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>44</td>
<td>No (0.639)</td>
<td>0.052</td>
<td>0.0003</td>
<td>9/22 (41%)</td>
<td>1/22 (5%)</td>
<td>12/22 (54%)</td>
</tr>
</tbody>
</table>

As in the analysis of Understandability Effectiveness, a more in-depth analysis of this measure was carried out by calculating the number of subjects who achieved better values when using the low or high LoD diagrams (i.e., a low LoD value is higher than a high LoD value or vice versa, respectively). As Table 3.12 shows, there were isolated cases (one or no cases in each experiment) with subjects who obtained the same \textit{UEffic} for both treatments (high and low LoD). More subjects performed better with a low LoD than with a high LoD in E-UL and R1-UCLM but the differences are very slight (1 subject and 3 subjects, respectively). In the case of R2-UB1 and R3-UB2, the results shows a slight tendency toward obtaining better results with a high LoD than with a low LoD (with a difference of 7 subjects in the case of R2-UB1).

The results suggest that there was no statistically significant difference in the \textit{UEffic} when the participants employed low or high LoD models. Bearing in mind that the statistical powers are very low, there is an 82-94.8% probability of wrongly assuming the equal influence of high and low level of detail.
3.4.2.3. Testing Modifiability Effectiveness (H3,0)

We tested the formulated hypothesis (H3,0) using a Wilcoxon test. Its results, which are shown in Table 3.13, reflect that the p-value is greater than 0.05 in the experiment and its replications, thus forcing us not to reject the hypothesis. This means that there is no significant difference in the MEffec in either group (low or high LoD diagram groups).

In the case of the MEffec, owing to the lack of significant results, we again calculated the number of subjects who performed better with low LoD diagrams than with high LoD diagrams and vice versa. The majority of subjects obtained better results with low LoD diagrams, with the exception of R3-UB2 in which the number of subjects who performed better with low LoD was the same as the subjects who performed better with high LoD. In the case of the MEffic, we can see a clear tendency toward obtaining better results with a low LoD, which is the opposite of what we had expected. The results suggest that there was no statistical significant difference in the MEffec when the participants employed low or high LoD models. Again, it is important to note the high probabilities of assuming a lack of influence of the level of detail owing to the low statistical powers.

Table 3.13. Wilcoxon tests results for MEffec.

<table>
<thead>
<tr>
<th>Dep. Variable</th>
<th>#obs</th>
<th>Influence (low≠high)</th>
<th>Statistical Power</th>
<th>Effect size</th>
<th># of low&gt;high</th>
<th># of low=high</th>
<th># of low&lt;high</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>22</td>
<td>No (0.505)</td>
<td>0.069</td>
<td>0.009</td>
<td>6/11 (55%)</td>
<td>1/11 (9%)</td>
<td>4/11 (36%)</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>32</td>
<td>No (0.535)</td>
<td>0.178</td>
<td>0.006</td>
<td>11/16 (65%)</td>
<td>0/16 (0%)</td>
<td>5/16 (35%)</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>58</td>
<td>No (0.309)</td>
<td>0.088</td>
<td>0.006</td>
<td>16/29 (55%)</td>
<td>0/29 (0%)</td>
<td>13/29 (45%)</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>44</td>
<td>No (0.346)</td>
<td>0.074</td>
<td>0.004</td>
<td>11/22 (50%)</td>
<td>0/22 (0%)</td>
<td>11/22 (50%)</td>
</tr>
</tbody>
</table>

3.4.2.4. Testing Modifiability Efficiency (H4,0)

Finally, we tested the last null hypothesis formulated (H4,0). The results, which are shown in Table 3.14, suggest that the null hypothesis cannot be rejected because the p-value is greater than 0.05 in the experiment and its replications. This means that, again, there is no significant difference in the MEffic in either group (low or high LoD diagram groups). As in the case of the other three metrics, the statistical powers for the MEffic are very low, and this indicates a possible high Type II error.

Table 3.14. Wilcoxon tests results for MEffic.

<table>
<thead>
<tr>
<th>Dep. Variable</th>
<th>#obs</th>
<th>Influence (low≠high)</th>
<th>Statistical Power</th>
<th>Effect size</th>
<th># of low&gt;high</th>
<th># of low=high</th>
<th># of low&lt;high</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>22</td>
<td>No (0.477)</td>
<td>0.070</td>
<td>0.010</td>
<td>5/11 (45%)</td>
<td>0/11 (0%)</td>
<td>6/11 (55%)</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>32</td>
<td>No (0.379)</td>
<td>0.117</td>
<td>0.020</td>
<td>9/16 (38%)</td>
<td>0/16 (0%)</td>
<td>7/16 (62%)</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>62</td>
<td>No (0.922)</td>
<td>0.092</td>
<td>0.006</td>
<td>15/31 (49%)</td>
<td>0/31 (9%)</td>
<td>16/31 (51%)</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>44</td>
<td>No (0.848)</td>
<td>0.103</td>
<td>0.011</td>
<td>12/22 (54%)</td>
<td>1/22 (5%)</td>
<td>9/22 (41%)</td>
</tr>
</tbody>
</table>

As shows, no subjects obtained the same MEffic for both treatments (high and low LoD). There were two cases (E-UL and R2-UB1) in which more subjects performed better with a high LoD than with a low LoD, but the differences in comparison to the
opposite group are very small (1 subject). In the other two cases (R1-UCLM and R3-UB2), the subjects performed better with a low LoD.

3.4.2.5. Integrating the obtained results through meta-analysis

Since there are no significant results in the tests as regards the influence of the LoD, and the effect sizes are small, we therefore decided to integrate the results of the different studies using a meta-analysis. A meta-analysis is a set of statistical techniques with which to integrate the results of various studies in order to obtain a global effect of a factor on a dependent variable. In our case the factor is the LoD of UML diagrams, and we wish to know how this affects the understandability and modifiability of source code. This technique has been used for the same purpose in other families of experiments, such as those shown in (Cruz-Lemus et al., 2010; Hannay et al., 2009; Scanniello et al., 2014).

As measures may originate from different environments and may not be homogeneous, it is necessary to obtain a standardized measure of each one, and the measures used to estimate the global size effect of the factor must then be combined. For each dependent variable, we computed the mean value obtained by the participants when using a low LoD, minus the mean value they obtained with a high LoD. We used these values to compute the Hedges’ g metric (Hedges and Olkin, 1985; Kampenes et al., 2007), which was used as a standardized measure. The overall conclusion was obtained by calculating the Z score based on the mean and standard deviation of the Hedges’ g statistics of the experiments. The global effect size was therefore obtained by using the Hedges’ g metric, in which the weights are proportional to the size of the experiment:

\[ \bar{Z} = \frac{\sum_i w_i z_i}{\sum_i w_i} \]

where \( w_i = 1/(n_i-3) \) and \( n_i \) is the sample size of the \( i \)-th experiment. The higher the value of Hedges’ g, the higher the corresponding mean difference. An effect size of 0.5 indicates that the mean value obtained when using low LoD diagrams is half a standard deviation greater than the mean when not using them.

As suggested in (Kampenes et al., 2007), the effect size can be classified as: small (S) for values between 0 and 0.37, medium (M) for values between 0.38 and 1.0, and large (L) for values above 1.00.

The meta-analysis was performed by using the Comprehensive Meta-Analysis v2 tool (Biostat Inc., 2006). For each measure the tool produced the forest plots depicted in Figure 3.4, Figure 3.5, Figure 3.6 and Figure 3.7. The squares and diamonds are mostly proportional in size to each study’s weight under the fixed effect model (see the ‘Relative weight’ column). The squares show the individual effect size of each experiment and the diamond shows the global effect size. The values of the Hedges’ g metric are also reported. Positive values indicate that the use of low LoD diagrams improves the comprehensibility and modifiability of source code. Negative values mean that a high LoD is the improving treatment.
Chapter 3: Does the Level of Detail (LoD) of UML Diagrams Influence Software Maintenance?

If we focus on the results obtained for the $UEffec$ variable (see Figure 3.4), the total effect is in favor of using high LoD diagrams, but the global effect size obtained is not statistically significant since the p-value is not greater than 0.05. The results are similar in the case of $UEffic$ (see Figure 3.5). The values obtained for the Hedge’s $g$ metric (0.123 for $UEffec$ and 0.070 for $UEffic$) indicate a small size for the global effect.

If we focus on the results obtained for the $MEffec$ variable (see Figure 3.6), the total effect is in favor of using low LoD diagrams, but the global effect size obtained is not statistically significant since the p-value is not greater than 0.05. The results are similar in the case of $MEffic$ (see Figure 3.7). The values obtained for the Hedge’s $g$ metric (-0.162 for $MEffec$ and -0.082 for $MEffic$) indicate a small size for the global effect.

![Figure 3.4. Meta-analysis of $UEffec$.](image)

![Figure 3.5. Meta-analysis of $UEffic$.](image)
Chapter 3: Does the Level of Detail (LoD) of UML Diagrams Influence Software Maintenance?

3.4.3. Influence of system

An analysis of the interaction plots shown in Figure 3.8, Figure 3.9, Figure 3.10 and Figure 3.11 shows that interaction is not present when the lines of the diagram are more or less parallel, which occurs in some cases and is the ideal situation.

In the case of E-UL (first plot of each figure), if we focus on those diagrams that have crossed lines we can separate them into two groups. The first is related to the understandability (concretely, the UEffec diagram), and in this case the participants appear to achieve better results when using System B with a low LoD and System A with a high LoD. These results allow us to suggest that System A has sufficient information in the low LoD version to understand the system, and when more information is introduced it appears to become more complex from the point of view of understanding. The second is related to modifiability (concretely, the MEffec and
MEffic diagrams). The opposite of the understandability group occurs when using System B, and the participants appear to obtain better results when using high LoD diagrams. The result seems to get worse if we add details to the diagrams when using System A. These results allow us to suggest that System B may be more complex than System A.

In the case of the results of R1-UCLM (second plot of each figure), there is less system interaction because the lines of the diagram are more or less parallel, and this is the ideal situation. Only in the case of MEffic do the subjects appear to perform better when adding details to System B.

The analysis of the interaction plots of R2-UB1 (third plot of each figure) shows similar results in the case of the understandability to those obtained in E-UL (the participants seem to achieve better results when using System B with a low LoD and System A with a high LoD). But in the case of the modifiability, the results are the opposite of those obtained in E-UL (the participants seem to obtain better results when they are using high LoD diagrams).

In the last replication, i.e., R3-UB2 (fourth plot of each figure), there is not so much system interaction because in almost all cases the lines of the diagram are more or less parallel, and this is the ideal situation. Only in the case of MEffic do the subjects seem to perform better when adding details to System B. This pattern was also discovered in the results of R1-UCLM.

Upon focusing on measurements, the plots can be separated in two groups. The first is related to the understandability (Figure 3.8 and Figure 3.9), and in these cases the participants seems to achieve better results when using System B. The second is related to the modifiability (Figure 3.10 and Figure 3.11), and in these cases the participants seems to achieve better results when using System A.

![Figure 3.8. Interaction between LoD and System for UEffec.](image)

![Figure 3.9. Interaction between LoD and System for UEffic.](image)
These results allow us to suggest that the low LoD version of System A contains sufficient information to understand the system, and when more information is introduced it seems to become more complex from the point of view of the understandability. The opposite occurs when using System B. But if we focus on modifiability tasks, the participants appear to obtain better results when using high LoD diagrams and this result seems to get worse as details are added to the diagrams. The opposite occurs when using System A. These results allow us to suggest that System B seems to be more complex than System A.

![Image](image1.png)

**Figure 3.10. Interaction between LoD and System for MEffec.**

![Image](image2.png)

**Figure 3.11. Interaction between LoD and System for MEffic.**

### 3.4.4. Influence of order

In this subsection we analyze the hypothesis related to the order in which the material was presented to the subjects, in which the ideal result is to accept the null hypothesis:

$H_0$: The order in which the material was presented to the subjects has no influence as regards the measure. $H_1$: $\neg H_0$.

These hypotheses were tested by performing a Mann-Whitney test (see Table 3.15).

<table>
<thead>
<tr>
<th>Order</th>
<th>$U_{Effec}$</th>
<th>$UEffic$</th>
<th>$MEffec$</th>
<th>$MEffic$</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>1</td>
<td>0.105</td>
<td>0.223</td>
<td>0.341</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>0.166</td>
<td><strong>0.044</strong></td>
<td>0.691</td>
<td>0.258</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>0.227</td>
<td>0.478</td>
<td><strong>0.016</strong></td>
<td><strong>0.039</strong></td>
</tr>
<tr>
<td>R3-UB2</td>
<td>0.98</td>
<td>0.216</td>
<td>0.188</td>
<td>0.46</td>
</tr>
</tbody>
</table>
The results of the test revealed that the null hypothesis could not be rejected for all the experiments in the case of \( UEffec \). This result means that the participants in the second run did not obtain significantly greater or smaller differences when using a low or high LoD in \( UEffec \).

In the case of \( UEffic \), the results of the Mann-Whitney test revealed that \( H_0 \) could not be rejected for \( E-UL \), \( R2-UB1 \), and \( R3-UB2 \), i.e., the second run of these experiments did not obtain significantly greater differences in \( UEffic \). With regard to \( R1-UCLM \), the test revealed that the null hypothesis could be rejected: the participants in the second runs obtained significantly smaller differences in \( UEffic \).

If we focus on the measures related to modifiability, \( MEffec \) and \( MEffic \), the results of the Mann-Whitney test revealed that \( H_0 \) could not be rejected for \( E-UL \), \( R1-UCLM \), and \( R3-UB2 \), i.e., the second run of these experiments did not obtain significantly greater differences in \( MEffec \) or \( MEffic \). With regard to \( R2-UB1 \), the test revealed that the null hypothesis could be rejected since the participants in the second runs obtained significantly smaller differences in \( MEffec \) or \( MEffic \).

### 3.4.5. Influence of ability

In this subsection, we analyze the hypothesis related to the subject’s ability, in which the ideal result is to accept the null hypothesis:

\[
H_0: \text{The subjects’ abilities have no influence on the measure. } H_1: \neg H_0.
\]

These hypotheses were tested by performing Mann-Whitney tests (see Table 3.16).

If we focus on the results of the test performed for \( E-UL \), they signify that the null hypothesis could not be rejected for half of the variables, and the subjects’ abilities thus influence the results of the experiment for half of the measures. This was the case of \( UEffic \) and \( MEffic \), with which the test revealed that the null hypothesis could be rejected because the participants with higher abilities understood the system faster and performed better modifications. In the case of the other two measures (\( UEffec \) and \( MEffec \)) ability had no influence on their results.

In the remaining cases (\( R1-UCLM \), \( R2-UB1 \) and \( R3-UB2 \)), the results of the test indicate that the null hypothesis could not be rejected for any of the variables. The subjects’ abilities did not therefore influence the results of the experiment as we had expected.

<table>
<thead>
<tr>
<th>Ability</th>
<th>( UEffec )</th>
<th>( UEffic )</th>
<th>( MEffec )</th>
<th>( MEffic )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-UL</td>
<td>0.632</td>
<td>0.031</td>
<td>0.007</td>
<td>0.092</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>0.084</td>
<td>0.876</td>
<td>0.183</td>
<td>0.139</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>0.161</td>
<td>0.085</td>
<td>0.899</td>
<td>0.829</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>0.673</td>
<td>0.888</td>
<td>0.279</td>
<td>0.197</td>
</tr>
</tbody>
</table>
3.4.6. Post-experiment questionnaire results

With regard to the analysis of the post-questionnaire, it is important to bear in mind that the subjects responded to 2 post-questionnaires each (one after each run). The absolute numbers in this section might therefore be higher than the total number of subjects. In order to avoid confusion, we shall refer to the number of “cases” rather than the number of “subjects” when providing these kinds of numbers.

The analysis of the answers to the post-experiment questionnaire revealed that the time needed to carry out the comprehension and modification tasks was considered to be inappropriate (more time was needed), and that the subjects considered the tasks to be quite difficult (Figure 3.12), independently of the treatment received. The need for more time to perform the tasks and the consideration of the task as difficult may have arisen from the fact that the measurement of the time needed was derived from the pilot study, which was performed by PhD students, who probably had a higher ability and/or more experience than these Master’s students, signifying that the less experienced subjects needed more time.

We also asked them if they had had any problems when reading the diagrams made available to them. In this case, if a subject responded “completely agree” or “agree”, this indicates that s/he did not have any problems when reading the diagrams, while a response of “disagree” or “completely disagree” indicates that they did, and “neither” is a neutral response. As Figure 3.13 shows, most of the subjects did not have any problems when reading the diagrams (90 vs. 38) and more particularly, the low LoD diagrams caused less problems than the high LoD diagrams (17 vs. 21). This can be explained because the structural complexity of a software diagram affects its cognitive complexity (Cant et al., 1995).

Cognitive complexity refers to the mental burden that people (e.g. analysts, designers, developers, testers, maintainers, etc.) experience when building, validating, verifying or using diagrams. According to Systems Theory, the complexity of a system is based on the number of different types of elements and on the number of
different types of (dynamically changing) relationships between them (Pippenger, 1978). The structural complexity of a software diagram is thus determined by the elements of which it is composed, signifying that a low LoD diagram has a lower structural complexity than a high LoD diagram, and that its cognitive complexity is therefore also lower.

Moreover, some subjects (11) had difficulties when reading the source code (Figure 3.14), more or less in the same proportion for the low and high LoD group in the case of E-UL. A majority of subjects also had difficulties when reading the source code (25 vs. 20) in the case of R2-UB1, and most of them were subjects from the high
LoD diagrams group (14 vs. 11). In the rest of the replications, the majority of the subjects did not have any difficulties when reading the source code. A major proportion of subjects from the low LoD diagrams group in the case of R1-UCLM, and subjects from the high LoD diagrams group in the case of R3-UB2 had difficulties when reading the diagrams.

Finally, we also asked about the subjects’ perceptions of some of the items that appeared in the high LoD diagrams but did not appear in the low LoD diagrams. Figure 3.15 shows that high LoD elements seem to be appreciated by the subjects. With regard to the histograms in Figure 3.15, if a subject responds “completely agree” or “agree”, this indicates that s/he thinks that the element in the question was helpful, while a response of “disagree” or “completely disagree” indicates that the elements in the question are not helpful (“neither” is a neutral response).

**Figure 3.15. Subjects’ opinion of LoD.**

If we focus on the elements related to class diagrams (upper histograms) we can see that attributes are helpful in 100 cases (versus 9 cases in which the subjects do not believe them to be helpful). The same is true for the operations (102 cases vs. 4 cases). If we focus on the elements related to sequence diagrams (lower histograms) we can see that formal messages are more helpful (107) than natural language messages (2),
and the same can also be said for the appearance of parameters in messages (102 vs. 5). In all these cases the graphs show that the subjects who received the high LoD diagrams agree more with the helpfulness of those elements.

These results allow us to suggest that the low LoD version of System A contains sufficient information to understand the system, and when more information is introduced it seems to become more complex from the point of view of the understandability. The opposite occurs when using System B. But if we focus on modifiability tasks, the participants appear to obtain better results when using high LoD diagrams and this result seems to get worse as details are added to the diagrams. The opposite occurs when using System A. These results allow us to suggest that System B seems to be more complex than System A.

### 3.5. SUMMARY AND DISCUSSION OF THE DATA ANALYSIS

The main findings of the family of experiments presented in this paper, which are also illustrated in Table 3.17, are summarized below. We shall also discuss and attempt to find an explanation for the results obtained.

The first experiment (E-UL) was performed with 11 student subjects from the University of Leiden (The Netherlands). In this experiment we did not obtain conclusive results in favor of any of the treatments (low or high LoD). In general, the cofactors did not influence the results of the experiment. But the subjects’ Ability did influence some results (UEffic and MEffic), and the System seemed to slightly influence the results of the measures (UEffic, UEffic, MEffic and MEffic). The descriptive statistics showed a tendency in favor of using low LoD, contrary to our expectations since we believed that more details in a diagram would help maintainers to perform their daily tasks. Indeed, in the results of the post-experiment questionnaire the subjects with high LoD diagrams had problems when reading the diagrams. We then attempted to replicate the experiment in order to corroborate the tendency of this preliminary conclusion.

The first replication (R1-UCLM) was performed by a different group of students, concretely 16 students from the University of Castilla-La Mancha (Spain). We attempted to solve the influence of the system detected in the previous experiment by clarifying the statements of the assignments, since the E-UL subjects’ answers seemed to follow an incorrect pattern which might have been caused by the statements rather than by the system itself. In this case, the influence of this cofactor was almost nil, i.e., we cancelled the influence of the system as we expected, and Order influenced only one of our four measures (UEffic). Following the results obtained in the first experiment, the descriptive statistics of the first replication were in favor of a low LoD, but again we did not obtain conclusive results after performing the statistical test. However, contrary to the results of the E-UL, the responses to the post-experiment questionnaire of R1-UCLM show that the subjects had more problems when reading the low LoD diagrams, and we therefore decided to perform two more replications.
Table 3.17. Summary of results of the family of experiments.

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Descriptive statistics (in favour of...)</th>
<th>Influence of LoD</th>
<th>Influence of System</th>
<th>Influence of Order</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UEffic</td>
<td>UEffic</td>
<td>MEffic</td>
<td>MEffic</td>
</tr>
<tr>
<td>E-UL</td>
<td>-</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>R1-UCLM</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>R2-UB1</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>R3-UB2</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>H</td>
</tr>
</tbody>
</table>

**Legend**

Descriptive statistics:
- “L” = better results when using LOW rather than high LoD
- “H” = better results when using HIGH rather than low LoD
- “-” = no differences when using low or high LoD

Influence of LoD:
- x = hypothesis not rejected → There is no significant difference in the results when working with UML diagrams modeled using high or low LoD
- v = hypothesis rejected → There is a significant difference in the results when working with UML diagrams modeled using high or low LoD (expected value)

Influence of System:
- x = hypothesis not rejected → There is no significant difference in the results when working with system A or B (expected value)
- v = hypothesis rejected → There is a significant difference in the results when working with system A or B

Influence of Order:
- x = hypothesis not rejected → There is no significant difference in the results when receiving system A first and system B second, or vice versa (expected value)
- v = hypothesis rejected → There is a significant difference in the results when receiving system A first and system B second, or vice versa

In these last two replications (R2-UB1 and R3-UB2) we attempted to solve the problem of obtaining inconclusive results by increasing the number of subjects involved in the experiment, since this might help us to detect patterns in the responses and perform a more powerful statistical test. In this case, 32 and 22 students, respectively, from the University of Bari (Italy) were involved as subjects. In the second replication (R2-UB1) there were clear influences of the system and order,
which might have been caused by learning effects or indirect effects. Attempts were made to fix these possible misunderstandings in the third replication (R3-UB2), and the cofactors did not have any influence on the results. Again, the statistical test did not provide conclusive results about the influence of the different LoDs on the understanding and modifiability of a system. Surprising, and contrary to the results of the first experiment and the first replication, the descriptive statistics of these two replications were in favor of a high LoD. Contrary to these results, the subjects had more difficulties when reading the high LoD diagrams compared to the low LoD diagrams.

After the individual analysis, a meta-analysis was performed in order to integrate the results. Its results show a tendency in favor of high LoD diagrams for both measures related to the understandability of the system (UEffec and UEffic) and in favor of low LoD diagrams for both measures related to the modifiability of source code (MEffec and MEffic). But in both cases, the results are not clearly evident owing to the values of the p-values (always lower than 0.05). These results are aligned with those obtained in the descriptive statistics.

Although the results do not seem to be in favor of a specific LoD in each experiment or replication, we noticed that in the majority of the cases, the descriptive statistics were in favor of using high LoD diagrams to understand the system, and a low LoD to maintain it.

Indeed, we performed replications of the experiment to explain or corroborate the results obtained, but we were quite frustrated by the fact that results seemed to be incoherent and random, without a clear tendency. With the goal of finding and explanation for the results obtained we decided to obtain feedback from the students who had performed the experiments, and we also wished to know the point of view of practitioners.

One year after the execution of the experiments, the majority of the students who had been involved in their realization took another course with the same professors. Occasionally, during lessons, the students asked their professors about the results obtained from them in the experiment, what hypothesis had been tested, etc.

In order to try to explain the results of the experiment, and in order to teach the students about designing and analyzing experiments, a focus-group-session explaining these concepts was planned at each location. The professors explained the details of the design of the experiment to their students (which could not been explained before in order to not influence their performance), along with the results. The results obtained (no convincing effect) were also presented to the students. They attempted to provide some feedback about the possible reasons, and in all locations they agreed that, in some cases, they had not used the diagrams, and this is therefore a clear factor that influenced the lack of results. They argued that they had not used the UML because the systems came from a well-known domain, and because the tasks were not very complex - even though they were challenging to complete in the time available. It could be a risk to base our explanations on a focus group that came into existence one year after the execution of the experiment because the subjects might have forgotten
the main details, but the majority of the students agreed on their responses, and we therefore considered them to be valid. It is also important to highlight that some of the students involved in the experiments during that year occasionally asked their professors about the results of the experiment, so they were obviously interested in the topic and were therefore less prone to forget it.

Another interesting opportunity for discussing and attempting to understand the results obtained was at a meeting with some practitioners from SER & Practices (a small Italian company focused on software development and software engineering research). In SER & Practices a weekly team meeting is held between the Director of the company (Danilo Caivano, one of the authors) and the three area managers (Innovation, Production and Services, Infrastructure). At the end of the meeting the Director asked the managers for help in interpreting the results obtained as part of a second focus group. To fully understand the manager’s point of view it is opportune to point out that they use a SCRUM based process for software systems development and maintenance.

A summary of the main findings of a long discussion and brainstorming session is shown below:

- The systems were too small to effectively require the use of UML diagrams or other types of technical documentation.
- The application domains of the two systems are too simple and well known to require the use of very detailed diagrams for system understanding, especially for students because these systems are usually used in textbooks or for didactical purposes.
- The maintenance request was simple and well-focused and did not therefore imply a time consuming impact analysis or the need to study the analysis or design diagrams.
- Under the conditions described above, the source code is the best documentation and the greater the level of detail in the diagram, the more difficult it is to understand. This might be caused by the fact that the time spent studying diagrams may be greater than that required to understand the system (or that part of the system was directly impacted by the maintenance request) directly through code.

Furthermore, the managers explained the way in which they use the diagrams, according to certain rules. These rules helped us to corroborate our findings, and are the following:

- When the systems maintained are large and/or technically complex (i.e., they are the result of the integration of several components developed by using different programming languages or development frameworks, etc.) they usually use the high level design diagrams/documentation in order to roughly understand the system architecture and thus which components/subsystem they have to understand and modify. They then attempt to understand the selected components directly from source code with the help of the diagrams if necessary.
Chapter 3: Does the Level of Detail (LoD) of UML Diagrams Influence Software Maintenance?

- When the systems maintained are complex and difficult to understand (they gave the example of real time or embedded systems in which there are typically several functions/methods/class that are not traceable to end user functions) they prefer to use diagrams with a higher level of detail or the analysis diagrams/documentation in order to understand the system behavior. This signifies that, depending on the technical complexity/size of the system, the subjects did or did not use the design diagrams.

Finally we obtained a conclusion that we would not have been able to extract from the results of the experiment and which was obtained from occasional feedback from the 2 focus groups (students and practitioners): during the experiment it is possible that not all the subjects used the UML diagrams provided. We did not obtain the same observations from the subjects because they did not use the UML diagrams in the same way, and we were consequently unable to evaluate the LoD effects. The lack of use of the UML diagrams might have been for several reasons:

- Size of system and knowledge of the domain: The systems used were small systems and they came from well-known domains. In this case, the subjects did not need to understand the system because they already more or less knew what it was like. They only needed to navigate the source code to discover how it was implemented.

- Time constraints: We attempted to design a realistic experiment using actual systems. But the time for the experiment was very limited owing to the subjects’ availability, so the tasks needed to be sufficiently simple to be carried out during the time of the experiment. The low complexity of the tasks might have meant that the subjects did not need a detailed understanding of the systems.

It might appear that the experiment was incorrectly designed. Nevertheless, we performed a pilot study to test both the material and the behavior of possible subjects, the results of which were satisfactory and helped us to improve the design of the experiment. However, the people who performed the pilot study were skilled PhD students who had more experience and abilities as regards maintaining a system by following a methodological process, particularly a model-based process.

The fact that the subjects did not use the UML diagrams provided might reflect the current situation of industrial maintainers: documentation is not taken into account when under pressure and when there is a time constraint. What is more, when the domain is well-known, or the maintainers are already familiarized with it, they minimize the time taken to read the UML diagrams and they may work directly on the source code. These results agree with those obtained in (Scanniello et al., 2014).

3.6. IMPLICATIONS OF THE STUDY

One of the main implications of the results obtained in this family of experiments is that practitioners do not need to pay so much attention to the LoD of the diagrams used during maintenance tasks when the system under maintenance comes from a well-known domain. This can only be generalized in the context of maintenance projects for Java systems performed by novice software maintainers. From the
researcher perspective, the results of this family of experiments could help them to design other experiments, or complete families, taking into account the problems dealt with, such as how to deal with randomized result without a clear tendency and how to obtain an explanation for them.

We adopted a perspective-based approach (Nugroho and Chaudron, 2008) to judge the implications of our family of experiments. In particular, we based our discussion on the practitioner/consultant (simply practitioner in the following) and researcher perspectives. To this end, we took advantage of the checklists proposed in (Anda et al., 2006) in order to plan and report this family of experiments. The main findings of our study and their research and practical implications are summarized as follows:

- The use of low LoD diagrams is better in comparison to that of high LoD diagrams. This statement is also supported by Briand et al.’s framework (Briand et al., 2001a), which hypothesizes that high cognitive complexity (i.e., high LoD diagrams) will result in reduced understandability which impedes the analyzability, adaptability and flexibility of the diagram. This hypothesized relationship between structural complexity and external quality properties like understandability and modifiability has been repeatedly demonstrated (Poels and Dedene, 2000; Siau, 1999). This result is relevant from both the practitioner and the researcher perspectives. From the practitioner perspective, this result is relevant because it is useless to provide maintainers with additional information, coinciding with the results found in (Gravino et al., 2010). From the researcher perspective, it would be interesting to investigate whether variations in the context lead to different results and why these diagrams are not as useful as expected. With regard to these points, our work provides interesting insights resulting from an empirical investigation based on 81 participants. Although this might not perhaps be surprising, this study provides the bases for future investigations that may be better focused on how the UML supports software engineers in the maintenance phase.

- The use of UML diagrams does not increase the performance of maintenance operations in small systems when developers are familiar with the domain, and they might distract the participants while performing comprehension and modification tasks. This result is relevant for the researcher because it would be interesting to investigate why participants’ comprehension of source code (independently of their experience) does not improve comprehension when it is supplemented with UML diagrams (low or high LoD). A plausible justification for this result is that the traceability of the UML diagrams and the source code is reduced each time that the source code is updated without synchronization with the diagrams: the names of the entity may be different in the diagrams and source code, or the relationships between the diagrams and the source code may be changed and become more intricate in the source code than in the diagrams.

- The participants’ familiarity with the problem domain of a software system might affect the comprehensibility and the modifiability of the source code, both when using and not using UML analysis models. This result could be of interest to both the researcher and the practitioner.
When performing maintenance tasks, the source of information that the participants found more useful was the code. The participants thus perceived that UML diagrams could not benefit the execution of the tasks assigned. This result is relevant for the researcher since it might be interesting to investigate the motivation that guides a software engineer as regards trusting a source of information and how s/he exploits it to accomplish a maintenance task.

The UML diagrams selected in the two software systems were realistic for small-sized and well-known projects. Although we are not sure that the results achieved scale to real projects, they may be of interest to practitioners working on cases in which the documentation is incomplete (e.g., in lean development processes) and the maintenance operation is executed on a subset of the source code of the entire system.

The UML is widely used in the software industry (Cohen et al., 2004; Dobing and Parsons, 2006). The results obtained are therefore useful for all those companies that exploit this notation as support for software maintainers/developers when executing maintenance operations.

From a methodological perspective there are the following lessons: Firstly, this study benefitted greatly from introducing the post-experiment questionnaire in the replications of the experiment. This questionnaire was targeted toward clarifying some of the questions that arose after performing the first experiment. Secondly, the study showed that the subjects and conditions of a study trial should be very carefully chosen. In our case, the Ph.D. students’ behavior in the trial was clearly different from that of the M.Sc. students in the actual experiment.

3.7. THREATS TO VALIDITY

It is necessary to consider certain issues that may have threatened the validity of the experiment:

- **External validity**: External validity may be threatened when experiments are performed with students, and the representativeness of the subjects in comparison to software professionals may be doubtful. In spite of this, the tasks to be performed did not require high levels of industrial experience, so we believed that this experiment could be considered appropriate, as suggested in literature (Basili et al., 1999; Bruegge and Dutoit, 2010; Carver, 2010; Höst et al., 2000). Working with students also implies a set of advantages, such as the fact that the students’ prior knowledge is fairly homogeneous, there is the possible availability of a large number of participants (Verelst, 2004) and there is the chance to test experimental design and initial hypotheses (Sjøberg et al., 2005). An additional advantage of using students in experiments concerning understandability and modifiability is that the cognitive complexity of the objects under study is not hidden by the participants’ experience. As suggested in (Singer and Vinson, 2002), ethical issues were dealt with carefully. There was no information in the raw data that could allow a particular individual to be identified. The names of the students who participated in the experiments were annotated to provide them with extra points in
their courses when needed, but it was not possible to link their names to their responses. Informed consent and confidentiality are not required in these cases. The students’ participation might also have been biased because they were able to benefit from it (extra points, examples of exam exercises), but their performance was not biased since all the participants obtained the same benefits, independently of their responses.

- The participants’ lack of familiarity with the problem domain of a software system might affect the understandability and the modifiability of the source code, thus biasing the results since it might be an extra cognitive effort. For this reason, and owing to the time constraints, we decided to use well-known domains as part of the experiment. There are no threats related to the material used since the systems used were real, if small, and based on well-known domains. We attempted to perform an experiment by simulating real conditions, and the subjects were therefore provided with all the documentation of the system together, as a maintainer would receive all the documentation of the maintenance project together.

- **Internal validity:** The threats to the internal validity have been mitigated by the design of the experiment. Each group of subjects worked on the same systems in different orders. Nevertheless, there is still the risk that the subjects might have learned how to improve their performances from one session to the next. In all the experiments, the scores achieved by the participants were not significantly better in the second run (with the exception of R1-UCLM with UEffic and R2-UB1 with MEffic and MEffic). For each experiment, the internal validity threat was also mitigated owing to the fact that the participants had similar experience with the UML, software system modeling, and computer programming. Furthermore, all the participants found the material provided, the tasks, and the goals of the experiment clear, as the post-experiment survey questionnaire results showed. Another issue concerns the exchange of information among the participants. The participants were not allowed to communicate with each other. We prevented this by monitoring them both during the runs and during the break between the first and the second task. Moreover, the instrumentation was tested in a pilot study in order to check its validity. In addition, mortality threats were mitigated by offering the subjects extra points in their final marks. Another internal threat could be the influence of the language used in the documentation. English was used in the experiment that took place in The Netherlands, and this may have represented a threat owing to the fact that English is not the participants’ native language. Students at Dutch Universities are, however, obliged to pass a high level English exam in order to gain entrance to the university, and we do not therefore consider this to be a high threat. The experimental material used in E-UL was in English, but the supervisors at the experiment supported the native speakers and helped them when needed (e.g., in the translation of technical terms). In the replications, the subjects were given the documentation in their native languages, signifying that this threat was mitigated. A further threat might have been the translation process; however, the experimental material was translated by native Spanish/Italian speakers.
• **Conclusion validity**: Conclusion validity concerns the data collection, the reliability of the measurement, and the validity of the statistical tests. Statistical tests were used to reject the null hypotheses. We have explicitly mentioned and discussed cases in which non-significant differences were present. Conclusion validity might also be affected by the number of observations. Further replications on larger datasets are therefore required to confirm or contradict the results.

• **Construct validity**: Construct validity may be influenced by: 1) the measures used to obtain a quantitative evaluation of the subjects’ performance, 2) the comprehension questionnaires, 3) the maintenance tasks, and 4) the post-experiment questionnaire. We used a well-known and widely used measure to obtain a quantitative evaluation of comprehensibility and modifiability. The understanding/modification tasks were formulated to condition the subjects’ answers in favor of neither low nor High LoD. The measures used were selected to achieve a balance between the correctness and completeness of the answers. The questionnaires were defined to obtain sufficiently complex questions without them being too obvious, and they were formulated in a similar way. The post-experiment questionnaire was designed using standard forms and scales. Social threats (e.g., evaluation apprehension) have been avoided, since the students were not graded on the results obtained. Other possible threats to construct validity could be related to: the translation of the experimental material and social threats. The first kind of threat was reduced through the involvement of a native speaker to translate all the material used. Social threats (i.e., evaluation apprehension) were avoided since we did not grade the students on the results obtained in the experiments. As the subjects were provided with all the documentation of the system together, we were not able to control whether or not they used the diagrams. This problem could be solved by dividing the experiment into two different phases, the first of which would be related to the understanding of the system in which only the UML diagrams would be provided, and the second of which would be related to the modifiability of the system using the UML diagrams provided plus the source code. This hypothetical design would make it possible to control the factor related to the use or non-use of the diagrams, but the experiment would be less realistic because in real scenarios maintainers can access all the information when they need it.

### 3.8. CONCLUSIONS AND FUTURE WORK

Many software projects do not produce complete or detailed documentation owing to time constraints. Many maintenance projects are therefore performed by maintainers who have to understand a software system based on the source code and the existing UML diagrams. An alternative might be to use UML diagrams obtained by using a reverse engineering process starting from the source code that is available. In this latter case, the diagrams would be much more detailed and less abstract than is the case of forward designed diagrams.

The Level of Detail (LoD) presented in UML diagrams might therefore be an influential factor in software maintenance tasks, but it could be different depending on
the development approach used. For example, in a model-based approach, the optimal LoD is not clear, but in an MDD approach a higher LoD would appear to be better as regards generating a more complete source code. We thus decided to carry out a family of four experiments in order to investigate whether the use of low LoD UML diagrams supports novice software engineers when comprehending and modifying the source code of small systems in comparison to high LoD UML diagrams during model-centric maintenance. We have particularly focused on the perfective maintenance tasks carried out by individual maintainers.

This family consisted of one experiment and three replications, carried out with students from The Netherlands, Spain and Italy. We used controlled experiments because a number of confounding and uncontrollable factors may be present in real project settings. In real projects, it may be impossible to control factors such as learning and/or fatigue effects and to select specific tasks. Controlled experiments also reduce failure risks related to long term empirical investigations (as in our case). Although questions about the external validity (e.g., generalization to realistic comprehension tasks on object oriented source code) may arise, controlled experiments have often been conducted in the early steps of empirical investigations that have taken place over the years (e.g. (Arisholm et al., 2006)).

The descriptive statistics might lead us to assume that high LoD diagrams are more helpful when understanding a system in comparison to low LoD, while low LoD diagrams are more helpful when carrying out maintenance tasks. Although the results obtained in this family of experiments do not appear to have a clear tendency in favor of high or low LoD based on the statistical test performed, two focus groups allowed us to extract the conclusion that UML diagrams should not be used, independently of their LoD, in the context of novice software maintainers when maintaining the source code of small realistic systems from well-known domains.

The empirical evidence obtained from the family of experiments should be considered valid in the context of undergraduate/graduate students (considered as novice software maintainers) who are maintaining relatively simple systems related to well-known domains. The questionable utility of the UML in this context might be caused by the kind of systems modeled: maintainers do not need so much information about the system when performing maintenance tasks on small systems from well-known domains. In the case of novice maintainers modifying small and well-known domains, and considering the findings obtained, we would therefore recommend that companies follow a model-centric approach in order to improve the understanding of the system and source code but not invest too much in the maintenance of the documentation related to the UML diagrams alone.

Possible future directions for our current research are: (i) performing further studies considering realistic software systems related to larger and unknown domains to verify whether the findings obtained are still valid; (ii) replicating the study with practitioners; (iii) analyzing the effect of different UML notations (iv) studying the effect of different LoDs when performing other types of maintenance (for example, corrective maintenance); and (v) analyzing the effect of high or low LoD UML
diagrams in projects using other kinds of development methodologies (for example MDD).

As the research presented in this work is part of a long term research effort concerning the benefits of the UML in software maintenance, we have complemented the current study with a survey completed by practitioners in order to investigate the kinds of software systems maintained by companies, their characteristics (complexity, size, domains, etc.), whether or not (in industry) companies use UML diagrams during source code maintenance, and the benefits of using the UML that practitioners perceive (https://es.surveymonkey.com/s/software-maintenance). We plan to complete the survey analysis in the near future.

Acknowledgements: This research has been funded by the GEODAS project (GEODAS-BC (Ministerio de Economía y Competitividad and FEDER, TIN2012-37493-C03-01).
CHAPTER 4. ARE FORWARD DESIGNED OR REVERSE-ENGINEERED UML DIAGRAMS MORE HELPFUL FOR CODE MAINTENANCE?

Chapter 4: Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?

ABSTRACT

Context: Although various success stories of model-based approaches are reported in literature, there is still a significant resistance to model-based development in many software organizations because the UML is perceived to be expensive and not necessarily cost-effective. It is also important to gather empirical evidence in which context and under which conditions the UML makes or does not make a practical difference. Objective: Our objective is to provide empirical evidence as to which UML diagrams are more helpful during software maintenance: Forward Designed (FD) UML diagrams or Reverse Engineered (RE) UML diagrams. Method: We carried out a family of experiments which consisted of one experiment and two replications with a total of 169 Computer Science undergraduate students. Results: The individual data analysis and the meta-analysis conducted on the whole family, show a tendency in favor of FD diagrams and are significantly different as regards the effectiveness and efficiency of the subjects who participated and played the role of maintainers. The analysis of the qualitative data, collected using a post-experiment survey, reveals that the subjects did not consider RE diagrams helpful. Conclusions: Our findings show that there are some objective results (descriptive statistics or statistical tests) related to the maintenance effectiveness and efficiency in favor of the use of FD UML diagrams during software maintenance. Subjective opinions also lead us to recommend the use of UML diagrams (especially class diagrams) created during the design phase for software maintenance because they improve the understanding of the system in comparison with RE diagrams. Nevertheless, we can only assume that these results are valid in the context of Computer Science undergraduate students when working with small systems related to well-known domains, and other contexts should be explored in order to reaffirm the results in an industrial context by carrying out replications with professionals.

Keywords: Software Maintenance; UML Diagrams; Reverse Engineering; Maintainability; Family of Experiments; Controlled Experiment; Survey.

4.1. INTRODUCTION

The benefits of using software documentation to comprehend and modify source code have been widely studied (Abbes et al., 2011; de Souza et al., 2005; Tilley and Huang, 2003). The authors of (Tryggeseth, 1997) report that having documentation available during system maintenance reduces the time needed to understand how to perform maintenance tasks by approximately 20 percent.

UML (OMG, 2010) has become the de facto standard modeling notation used to complement software documentation as a graphical notation. It first appeared in 1997 and has now become one of the most widely-used modeling languages in industry. There is empirical evidence of its benefits during software development because it increases the understanding between customer and developer and improves communication among team members (Nugroho and Chaudron, 2009). UML also improves the source code quality by reducing its defect density (Nugroho et al., 2008).
Chapter 4: Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?

There is also some evidence of the UML’s benefits during software maintenance. The availability of UML documentation may result in significant improvements to the functional correctness of changes in addition to their design quality. However, it does not appear that any time is saved as a result. For simpler tasks, the time needed to update the UML documentation may be substantial in comparison with the potential benefits, thus motivating the need for UML tools with better support for software maintenance (Arisholm et al., 2006; Dzidek et al., 2008).

Despite these benefits, there are some common modeling problems that lead to models being less effective and less adopted (Berenbach and Konrad, 2007). For example, some of the limitations of program understating and maintenance supported by UML are the following:

- unclear specifications of syntax and semantics in some of the UML’s more advanced features (Tilley and Huang, 2003), leading to the need for adequate training in UML (Anda et al., 2006),
- spatial layout problems, e.g., large diagrams are not easy to read (Tilley and Huang, 2003),
- the UML’s insufficient support as regards representing the domain knowledge required to understand a program (Tilley and Huang, 2003),
- The time spent updating the UML diagrams according to the changes in source code counteracts the improvement in source code maintenance time (Arisholm et al., 2006; Dzidek et al., 2008).
- UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability (Scanniello et al., 2014).

Despite its limitations, there are clear benefits of using UML, such as the improvement in the traceability from functional requirements to code (Anda et al., 2006), and the fact that UML is always beneficial in terms of functional correctness (reducing defect density) (Nugroho and Chaudron, 2007).

Moreover, diagrams with a high level of detail are reported to be more helpful during software development (Nugroho, 2009), while those with a low level of detail seems to be better when performing maintenance tasks (Fernández-Sáez et al., 2012).

As we can see, there is some evidence to support the use of UML modeling during software maintenance, but the results are not conclusive owing to the diversity of the results obtained in different empirical studies. It is not clear whether an investment in creating and updating UML diagrams has a return during later phases, like software maintenance which consumes a high percentage of a project’s resources. If the presence of UML diagrams is a clear positive factor, but it is not clear what the best way to create them is, then why not create them directly from source code and save the time needed for their creation and update? This is related to a possible influential factor in the software maintenance which has not being studied before: the Origin of the diagrams. FD diagrams, i.e., the diagrams generated during forward development, are sometimes available for maintainers in the maintenance phase, but when this is not the case, the diagrams may be reconstructed through an RE technique (Perez-Castillo...
The difference in the origin of the diagrams (i.e., FD diagrams or RE diagrams) and the different techniques that can be used to generate an RE diagram result in different styles of diagrams that may influence the quality of the source code being maintained.

RE diagrams are easier to obtain than FD diagrams because they can be automatically generated with a reversing tool without high investments in developer effort (although these kinds of diagrams need some polishing). Given the ease of their generation and that they can be generated automatically at any time, maintainers can have up-to-date diagrams that model the system when they need them. However, the problem with these diagrams is their very high level of detail, which may lead to difficulties in understanding them. There are several issues related to the obtaining of diagrams with a high level of detail when they originate from source code, after applying a reverse engineering technique:

- The level of abstraction is very low, owing to the fact that every element in the source code is represented in the UML diagrams. The benefit of this is that there is a very high traceability from the diagrams to the source code.
- The business rules allow the designers to create UML diagrams by following a specific design objective. The developers then implement the source code by following these diagrams. RE diagrams do not represent these rules, since that they are obtained from source code and these diagrams only reflect how the code was implemented, rather than why.
- These RE diagrams, unlike FD diagrams, are platform-dependent and therefore contain details about the implementation patterns and frameworks used, which do not appear in FD diagrams.
- After obtaining the RE diagrams, a cleaning and lay-outing process needs to be performed in order to adapt them to their corresponding audience.

However, there is another option when up-to-date diagrams are required: the maintainer may keep the source code and the diagrams in-synch manually by applying the corresponding changes incurred by maintenance to both. This option requires more manual effort than the RE process, because the process is not as automated as the RE approach is. Nevertheless, when the diagrams are generated by people and not by automated tools, they may contain different levels of abstraction and detail, depending on the importance of diagram elements (Osman et al., 2012); this may make diagrams more understandable, and hence more effective.

All of the above lead us to pose our main research question: “Should software maintenance companies spend time updating their UML diagrams or should they use Reverse Engineered diagrams instead?” Our results might be useful for companies that are performing software maintenance and yet are unsure whether they should continue updating their UML diagrams (as part of the project documentation) or whether they might save that time by automatically generating RE diagrams.

On the one hand, if we obtain better results with design UML diagrams we will have empirical evidence to encourage companies and software developers to follow a
model-centric approach. This implies beginning the development of a software system by building the corresponding UML diagrams and keeping them up-to-date, thereby facilitating maintenance tasks. On the other hand, if we obtain better results with RE diagrams, we will have empirical evidence to suggest that maintainers should obtain the UML diagrams needed by using RE techniques. This will thus avoid the need to maintain the available diagrams (whenever these are available) and will reduce the time involved in maintenance tasks.

There is a need for experiments focused on software maintenance and evolution by a non-original developer, as this consumes the majority of the resources in a typical software organization.

All of the above led us to perform a family of experiments to investigate whether the different Origins of UML diagrams (Reverse Engineered or Forward Design diagrams) affect the maintainer’s performance when modifying source code. Our aim is to discover whether or not, in order to obtain an up-to-date version of diagrams, an effort should be made to maintain diagrams. The family consists of a controlled experiment, which was previously presented in (Fernández-Sáez et al., 2013b), and two replications, carried out with students from two countries (Spain, and Italy). The participants were 169 Computer Science undergraduate students with different abilities and levels of experience with the UML and with JAVA source code.

The three main goals of this paper are therefore to present a thorough description of this family of experiments, its main findings and their practical implications.

This paper is organized as follows. Section 4.2 presents the related work. Section 4.3 provides a description of the experiment and replications. The results obtained in the experiment are set out in Section 4.4, whilst the implications of the study are summarized in Section 4.5 and the threats to validity are summarized in Section 4.6. Finally, Section 4.7 outlines our main conclusions and future work.

4.2. RELATED WORK

The related work will be focused on the empirical evidence on the use of UML diagrams in software maintenance. A Systematic Literature Review that was carried out to collect all the empirical studies performed as regards the use of UML in maintenance and the understandability of UML diagrams which may influence the maintenance of the system is presented in (Fernández-Sáez et al., 2013a). In this SLR, 46 primary studies were found reporting 74 empirical studies related to this topic, but only the following works directly related to the use of UML diagrams in source code maintenance:

- In (Dzidek et al., 2008) an experiment was performed to investigate whether the use of UML influences maintenance in comparison to the use of only source code. This experiment investigates the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real nontrivial system, using 20 professional developers as subjects. These maintainers had to perform 5 maintenance tasks consisting of adding new functionalities to the
existing system after which the correctness, time and quality of the solution were measured. Both source code and UML diagrams, when available, had to be maintained. The results of this work show a positive influence of the presence of UML for maintainers. In terms of time, the UML subjects took more time if the UML documentation had to be updated but that difference was not statistically significant. However, UML was always beneficial in terms of functional correctness (introducing fewer faults into the software) because the subjects in the UML group had, on average, a practically and statistically significant 54 percent increase in the functional correctness of changes. UML also helped produce code of a better quality when the developers were not yet familiar with the system. This experiment is a replication of a previous work performed with students which is presented in (Arisholm et al., 2006) and which obtained similar results. The main difference between (Dzidek et al., 2008) and the work presented here is that (Dzidek et al., 2008) shows how source code is maintained when UML diagrams are complementing the source code or when the source code is alone, while the subjects of the family of experiments presented herein always received UML diagrams (some received FD diagrams while others received RE diagrams). Another difference is that the UML diagrams used in our family of experiments did not need to be updated according to source code changes. In addition, the subjects of (Dzidek et al., 2008) were professionals.

• Arisholm et al. (Arisholm et al., 2006) presented the results of two controlled experiments carried out to assess the impact of UML design diagrams on software maintenance. 98 undergraduate students were involved. The authors analyzed the time taken to perform the modifications to the system, the time spent on maintaining the models, and the quality of the modifications performed. The results of the quantitative analysis revealed no significant difference in the time spent making the modifications. Similarly to (Dzidek et al., 2008), they observed that the quality of the modifications was higher for those participants who were furnished with UML diagrams. As in (Dzidek et al., 2008), the participants’ ability and experience were not analyzed with regard to the comprehensibility and modifiability of source code. Unlike our study, the authors analyzed the effect of UML based documentation (a use case diagram, sequence diagrams for each use case, and a class diagram) on modification tasks performed on both UML diagrams and source code. The differences with the work presented in (Arisholm et al., 2006) and our work are similar to those mentioned in the case of (Dzidek et al., 2008), except for the type of subjects involved.

It is important to mention some other important papers relating to the influence of the use of UML diagrams during software maintenance which were not found as part of (Fernández-Sáez et al., 2013a) because of their dates of publication or because of the objective of the SLR:

• Scanniello et al. (Scanniello et al., 2014) used a family of controlled experiments to discover that the use of analysis UML diagrams (those obtained in an early phase
or the development process, such as the requirements elicitation or analysis phase) does not significantly improve the comprehension and modifiability of source code with regard to the use of source code alone. These results are valid in the context of undergraduate students and small size systems related to well-known domains. The study presented herein is, however, focused on FD and RE diagrams, which have a higher level of detail. Moreover, the dependent variables are different because the work of Scanniello et al. focused on the comprehension and maintainability of the source code, while the work presented here is focused solely on the maintainability of source code, and bigger systems are also considered.

- In (Scanniello et al., 2012) the results of an experiment to assess whether the comprehension of source code is affected when it is added to UML class and sequence diagrams produced in the design phase is presented. The results reveal that the participants benefited from the use of the UML diagrams. An average improvement of 14% was achieved when the participants accomplished the comprehension task with the class and sequence diagrams, but the time needed to comprehend the code was not significantly influenced from a statistical point of view. Our work differs from that presented in (Scanniello et al., 2012) because we compare FD diagrams with RE diagrams rather than with no UML diagrams, and the dependent variable is also different because (Scanniello et al., 2012) focuses on the comprehension of the source code and the work presented here is focused on the maintainability of source code. Bigger systems are also used here.

- In the work presented in (Karahasanovic and Thomas, 2007), the experiment performed is focused on the comprehension and the difficulties involved in maintaining object-oriented systems. During this experiment the subjects, who were 34 students in their third year of a computer science degree, had to perform 3 different maintenance tasks on a medium-size object-oriented system written in Java. The tasks were related to extending, updating and deleting functionalities of the system, i.e., maintaining the system. It is important to note that the subjects had to give higher priority to the quality of the solutions than to a shorter development time, which may have influenced the results of the experiment. UML diagrams were also presented to the subjects of the experiment, and the correctness of their solutions was measured, but this work was only focused on exploring the participants’ strategies and problems while they were conducting maintenance tasks on an object-oriented application. Two major groups of difficulties were related to the comprehension of the application structure, namely the understanding of GUI implementation and OO comprehension and programming. The main difference between the work presented in (Karahasanovic and Thomas, 2007) and the work here is that the subjects’ performance was not measured in the former while it is measured in the latter. In (Karahasanovic and Thomas, 2007) the results are qualitative (description of problems while maintaining source code), while here they are quantitative (results on performance, such as time spent and correctness of the answer).
• The authors of (Scanniello et al., 2010) show the results of an explorative survey used to investigate the state of the practice regarding the use of UML in software development and maintenance. The majority of the companies interviewed use UML for software development and to perform maintenance operations. Maintenance operations are mainly performed by practitioners with little experience. Another interesting point concerns the average effort needed to perform maintenance operations, which ranges from 1 to 5 person hours for an ordinary maintenance operation (e.g., corrective changes), and from 10 to 50 person hours in the case of an extraordinary maintenance operation (e.g., perfective or adaptive changes). The differences between the work presented in (Scanniello et al., 2010) and ours are based on the nature of the empirical study (an explorative survey vs. an explicative family of experiments).

• An experiment presented in (Fernández-Sáez et al., 2012) studies whether different Levels of Detail (LoD) in UML diagrams might influence the maintenance of source code. In (Fernández-Sáez et al., 2012; Nugroho, 2009) there is an assumption that the higher the amount of information put into a diagram, the more is known about the concepts/knowledge described in it. That being the case, a higher LoD would improve maintainers’ performances owing to the fact that they would understand the system they have to maintain better. The results from (Fernández-Sáez et al., 2012) are not conclusive, but show a slight tendency in favor of high LoD diagrams. The similarities of the work presented in this paper with (Fernández-Sáez et al., 2012) arise from the fact that FD diagrams could be considered as having a high LoD, while RE diagrams could be considered as having a very high LoD. The difference between (Fernández-Sáez et al., 2012) and the work presented here are the independent variables used (LoD of UML diagrams in the former, and the origin of UML diagrams in the latter), and the design of the experiment (within-subjects and between-subjects respectively).

Having assumed that the presence of UML diagrams is a positive factor for software maintenance, it is important to know what kinds of UML diagrams are better at improving these kinds of tasks. It is therefore important to additionally study the comprehension of UML diagram in themselves. It is possible to find many papers related to the comprehension of UML diagrams (which is directly related to the comprehension of the software system) in literature (Dobing and Parsons, 2006; Genero et al., 2011). We can highlight two papers found as part of (Fernández-Sáez et al., 2013b), which focus solely on software development using different kinds of UML diagrams during this phase but which might also be related to maintenance:

• The study presented in (Nugroho, 2009) consists of an empirical experiment focused on determining how different Levels of Detail (LoD) influence the understandability of UML diagrams. It measures the correctness and efficiency in comprehending UML diagrams of 53 Computer Science Master’s students. The results show a better understanding of diagrams when they have a high LoD. There are some differences between the work presented in (Nugroho, 2009) and the
family of experiments summarized in this paper. On the one hand, the experiment presented in (Nugroho, 2009) focuses solely on the comprehension of UML diagrams, and the subjects were not given the source code of the system. The experiment presented in (Nugroho, 2009) is similar to that presented in (Fernández-Sáez et al., 2012), but focuses on the development of software rather than the maintenance of software and with the difference that (Nugroho, 2009) is focused on the comprehension of the UML diagrams and (Fernández-Sáez et al., 2012) on the maintenance of source code.

• In (Tilley and Huang, 2003) an experiment with 15 subjects (PhD students or Professors) to assess the qualitative efficacy of UML diagrams in aiding program understanding is described. The experiment had participants analyze a series of UML diagrams and answer a detailed questionnaire concerning a hypothetical software system. Results from the experiment suggest that the relation between the correctness of the solution and the time spent obtaining it using UML to support program understanding is limited by factors such as ill-defined syntax and semantics, spatial layout, and domain knowledge.

All related work that is relevant to this paper is summarized in Table 4.1 to provide the reader with a better extract of the main information in order to compare the empirical studies. The columns of the table are described as follows:

• Ref: contains the reference to the paper that presents the empirical study considered.
• Type of empirical study: indicates the type of empirical study summarized in the paper (a survey, an experiment, a family of experiments, etc.)
• Goal: describes the goal pursued by the empirical study.
• Subjects: presents the numbers of subjects who participated in the empirical studies and the type of subjects (students, professionals, academic staff, etc.).
• Independent variables: describes the variables that are studied to ascertain their effect on the dependent variables. The values (treatments) of the independent variables are also presented.
• Dependent variables: presents the outcome variables, which are the variables that are affected by the changes produced in the independent variables.
• Experiment design: contains the type of design selected, which can be between-subjects (each subject receives only one treatment) or within subjects (each subject receives all the treatments).
• Tasks: describes the tasks to be performed by the subjects as part of the empirical study.
• Results: reveals the main findings obtained.
### Table 4.1. Summary of the related work (part 1/2).

<table>
<thead>
<tr>
<th>Ref</th>
<th>Study</th>
<th>Goal</th>
<th>Subjects</th>
<th>Independent variables</th>
<th>Dependent variables</th>
<th>Design</th>
<th>Tasks</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Dzidek et al., 2008)</td>
<td>1 experiment</td>
<td>Investigates whether the use of UML influences maintenance in comparison to use of only source code</td>
<td>20 professional developers</td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code)</td>
<td>Time spent modifying source code. Time spent modifying source code + UML diagrams. Functional correctness and quality of the solution.</td>
<td>Between-subjects</td>
<td>Modify source code and UML diagrams</td>
<td>The subjects who received UML diagrams needed more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce code of a better quality when the developers were not yet familiar with the system.</td>
</tr>
<tr>
<td>(Arisholm et al., 2006)</td>
<td>2 experiments</td>
<td>Investigates whether the use of UML influences maintenance in comparison to the use of only source code</td>
<td>Undergraduate students (22 and 76, respectively)</td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code)</td>
<td>Time needed to change source code. Time needed to change source code + UML diagrams. Correctness of the change. Quality of the change.</td>
<td>Between-subjects</td>
<td>Modify source code and UML diagrams</td>
<td>The subjects who received UML diagrams needed more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce code of a better quality when the developers were not yet familiar with the system.</td>
</tr>
<tr>
<td>(Scanniello et al., 2014)</td>
<td>Family of experiments (1+3)</td>
<td>Investigates whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code.</td>
<td>Undergraduate students (24, 22, 22 and 18)</td>
<td>The use of UML analysis models (possible values: presence or absence of UML diagrams accompanying source code)</td>
<td>Comprehension of level of the source code. Capability of a maintainer to modify source code.</td>
<td>Within-subjects</td>
<td>Comprehension and modification tasks. Post-experiment questions</td>
<td>UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.</td>
</tr>
<tr>
<td>(Scanniello et al., 2012)</td>
<td>1 experiment</td>
<td>Investigates whether the comprehension of source code increases when participants are provided with UML class and sequence diagrams produced in the software design phase</td>
<td>16 undergraduate students</td>
<td>The use of sequence and class diagrams created in the design phase (possible values: presence or absence of UML diagrams accompanying source code)</td>
<td>Comprehension of source code.</td>
<td>Within-subjects</td>
<td>Comprehension questions</td>
<td>Participants comprehend source code significantly better when it is added to class and sequence diagrams together.</td>
</tr>
</tbody>
</table>
Table 4.1. Summary of the related work (part 2/2).

<table>
<thead>
<tr>
<th>Ref</th>
<th>Type of study</th>
<th>Goal</th>
<th>Subjects</th>
<th>Independent variables</th>
<th>Dependent variables</th>
<th>Design</th>
<th>Tasks</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Karahasanovic and Thomas, 2007)</td>
<td>1 experiment</td>
<td>Investigates the comprehension and the difficulties involved in maintaining object-oriented systems</td>
<td>34 undergraduate students</td>
<td>-</td>
<td>Comprehension and modification of systems</td>
<td>Between-subjects</td>
<td>Modification questions</td>
<td>Detection of common difficulties in understanding when maintaining software systems.</td>
</tr>
<tr>
<td>(Scanniello et al., 2010)</td>
<td>1 survey</td>
<td>Investigates the state of the practice regarding the use of UML in software development and maintenance in Italian industry</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Between-subjects</td>
<td>-</td>
<td>The majority of the companies interviewed use UML for software development and to perform maintenance operations. Maintenance operations are mainly performed by practitioners with little experience.</td>
</tr>
<tr>
<td>(Fernández-Sáez et al., 2012)</td>
<td>1 experiment</td>
<td>Determines the influence of different levels of detail (LoD) of UML diagrams in source code maintenance</td>
<td>11 undergraduate students</td>
<td>-</td>
<td>Understandability and modifiability of source code</td>
<td>Between-subjects</td>
<td>Comprehension tasks + modification tasks + subjective questions</td>
<td>No significant results in favor of high or low LoD. There is a slight tendency in favor of low LoD diagrams</td>
</tr>
<tr>
<td>(Nugroho, 2009)</td>
<td>1 experiment</td>
<td>Determines the influence of different levels of detail (LoD) on the understandability of UML diagrams</td>
<td>-</td>
<td>-</td>
<td>Comprehension of UML diagrams</td>
<td>Between-subjects</td>
<td>Comprehension tasks</td>
<td>Results show that the effect of LoD in UML models on model comprehension is significant in favor of high LoD</td>
</tr>
<tr>
<td>(Tilley and Huang, 2003)</td>
<td>1 experiment</td>
<td>Assesses the qualitative efficacy of UML diagrams in aiding program understanding</td>
<td>54 Master’s students</td>
<td>-</td>
<td>Comprehension of UML diagrams</td>
<td>Between-subjects</td>
<td>-</td>
<td>UML’s efficacy in support of program understanding is limited by factors such as ill-defined syntax and semantics, spatial layout, and domain knowledge</td>
</tr>
</tbody>
</table>
It is clear that the LoD of UML diagrams is an important factor which has previously been studied in literature because the LoD of UML diagrams influences their understanding (and hence also source code understanding), and adding details to a diagram is a time consuming task. On the one hand, low LoD diagrams, like analysis models, do not appear to be helpful for source code maintenance (Scanniello et al., 2014). On the other hand, high LoD diagrams are better understood than low LoD diagrams when performing software development (Nugroho, 2009). It is therefore clear that a balanced LoD will be the best option, but there is a tendency to use diagrams with a detailed designed, especially when FD are used (Scanniello et al., 2012). But how many details should be presented in a UML diagram? If a high LoD diagram is the best option, why not use an RE diagram, which can be automatically generated (saving time) in comparison to creating UML diagrams during software development and keeping them up-to-date? These questions motivated us to plan a family of experiments that compare the usefulness of FD and RE diagrams when performing source code maintenance.

4.3. DESCRIPTION OF EXPERIMENT

Families of experiments allow researchers to answer questions that are beyond the scope of individual experiments and permit the generalization of findings across studies, thus providing evidence with which to for confirm or reject specific hypotheses (Basili et al., 1999). Replications of empirical studies might be regarded as an essential activity in the construction of knowledge in any empirical science based on the following propositions: “We do not take even our own observations quite seriously, or accept them as scientific observations, until we have repeated and tested them” (Popper, 1959) and “… replication is needed not merely to validate one’s findings, but more importantly, to establish the increasing range of radically different conditions under which the findings hold, and the predictable exceptions”.

In order to run and report this family of experiments, we followed the recommendations provided in several pieces of work (Jedlitschka et al., 2008; Juristo and Moreno, 2001; Wohlin et al., 2012). The family of experiments followed the guidelines for reporting empirical research in software engineering (Jedlitschka et al., 2008) as closely as possible. The experimental material is available for downloading at: http://alarcos.esi.uclm.es/originUMLmaintenance/

In the following subsections we shall describe the main characteristics of the experiment and the replications, including goal, context, variables, subjects, design, hypotheses, material, tasks, experiment procedure and analysis procedure. Figure 4.1 presents the chronology of the family of experiments, summarizing the name of the experiment, the number of participants, and the name of the Universities at which the experiment and replications were run.
4.3.1. Goal

The principal goal of this family of experiments was to investigate whether the Origin of UML diagrams influences the maintenance of source code. The GQM template for goal definition (Basili and Weiss, 1984; Basili et al., 1999) was used to define the goal of our experiment as follows: “Analyze the maintainability of source code from the point of view of software maintainers with regard to the Origin of the UML diagrams, in the context of Computer Science students at the Universities of Seville and Bari”.

We considered two possible Origins of the diagrams: the design phase and an RE technique. In the first case, our intention was to maintain the source code using the UML diagrams built during the design phase. In the second case, we set out to maintain a source code for which the UML diagrams were not available, which meant that they would have to be obtained from the source code using an RE technique.

We decided to consider class diagrams and sequence diagrams because they can be obtained from an RE technique and because they are also two of the most commonly used diagrams when designing a system (Dobing and Parsons, 2006; Erickson and Siau, 2007; Grossman et al., 2005).

4.3.2. Context selection

The experimental objects consisted of class and sequence diagrams and the Java code of one system. The diagrams were obtained from different Origins:

- **RE**: Reverse Engineered UML diagrams, which are constructed completely automatically based on the source code.
- **FD**: Forward designed UML diagrams obtained at the design phase. These are totally manually designed.

RE diagrams are diagrams with a high LoD since they represent all the elements in the source code. The FD diagrams might also be considered as diagrams with a high LoD because their class diagrams contain class names, attributes, operations and relationships, and their sequence diagrams contain lifelines, messages and parameters of messages. However, FD diagrams do not represent all the elements in the source code.
code, but those elements which are represented (based on human selection) are completely represented. FD diagrams can therefore be considered as high LoD diagrams while RE are higher LoD diagrams.

The diagrams described a sports center system from which users can rent services (tennis courts, etc.). The system is a sports center application which was created as part of the Master’s degree Thesis of a student from the University of Castilla-La Mancha, and we therefore consider it to be a realistic system. It is a desktop application created with the client-server paradigm. The system contains 5123 Lines of Code (LoC) (Table 4.2), so it might be considered a small realistic system. In fact its size is almost double the LoC of other systems used in previous works which have nevertheless been considered as realistic systems, for example in (Dzidek et al., 2008).

The maintenance requirements were formulated by the Master’s supervisor. In the case of the FD diagrams, 4 class diagrams are available, with a total of 16 classes, and 21 sequence diagrams, with 226 messages. In the case of RE diagrams, 4 class diagrams are available, with 21 classes, and 11 sequence diagrams, with 191 messages. The number of classes in class diagrams is a good deal smaller than in the class diagrams of systems used in other previous work. This is owing to the use of different levels of abstraction for modeling, but their diagram size is still representative of realistic systems (Heijstek and Chaudron, 2009). Note that the number of sequence diagrams in the RE group is 11 and the number of diagrams in D group is 21. The number of messages per diagram (226 messages for 21 diagrams in D group, and 191 messages for 11 diagrams in RE group) therefore provides us with an indicator which suggests that RE diagrams should be considered as being larger and more complex. The RE diagrams were generated using the IBM Rational Software Architect tool, employing the default RE-functionality that it provides, followed by auto-layouting (also offered by the same tool). These experimental objects were presented in Spanish in E-US1 and R-US2. A native Italian speaker translated all the material into Italian in the case of R-UB. The replicators supported the native speakers and helped them when needed (e.g., in the translation of technical terms).

<table>
<thead>
<tr>
<th>#Class diagrams</th>
<th>#classes</th>
<th>#Sequence diagrams</th>
<th>#messages</th>
<th>LoC</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>4</td>
<td>16</td>
<td>21</td>
<td>226</td>
</tr>
<tr>
<td>RE</td>
<td>4</td>
<td>21</td>
<td>11</td>
<td>191</td>
</tr>
</tbody>
</table>

The participants, who were grouped by experiment, had the following characteristics:

- **E-US1**: The participants were 40 Computer Science students from the University of Seville (Spain) who were taking the Software Engineering III course in the second-year of their Master’s Degree, from which they had acquired training in UML diagrams (as they also had from previous Software Engineering courses). 90% of the students were non-repeating course students and most of them did not have industrial experience (five subjects had experience as programmers, another as an analyst and another as a maintainer). The subjects considered their...
knowledge of the UML to be high/very high in most cases for class diagrams, and medium/high for sequence diagrams (the two kinds of diagrams which were used during the experiment). Most of the subjects’ knowledge of JAVA language is excellent (all the subjects considered it to be above average). In Spain, grades are expressed as doubles and assume values of between 5 and 10. The lowest grade is 5, while the highest is 10.

- **R-US2**: The participants were 51 students from the University of Seville (Spain) who were taking the same course as the students in the original experiment, i.e., they were taking the Software Engineering III course in the second-year of their Master’s Degree, from which they had acquired training in UML diagrams (as they also had from previous Software Engineering courses). In this case, more than half (58%) the students were repeating the course and most of them had industrial experience, mainly as programmers, but also as analysts, designers or maintainers (40% of the subjects did not have any experience at all). The subjects considered their knowledge of the UML to be high/very high in most cases for class diagrams and for sequence diagrams, and most of them had an excellent knowledge of JAVA language (all the subjects considered this to be above average). The grading system is analogous to that explained for E-US1.

- **R-UB**: The participants were 78 Computer Science students who were taking the Software Engineering course in the second-year of their Laurea Degree at the University of Bari. In Italy, the exam grades are expressed as integers and assume values of between 18 and 30. The lowest grade is 18, while the highest is 30. We cannot provide details about their background because the first session of the experiment needed to be canceled for reasons beyond our control.

The students who participated in the experiment and the replications were volunteers selected for convenience (the students available in the corresponding course).

### 4.3.3. Variable selection

The independent variable (also called the “main factor”) is the *Origin* of diagrams, which is a nominal variable with two values (treatments):

- Forward Designed (FD)
- Reverse Engineered (RE).

The dependent variable is the maintainability. This dependent variable was measured using the following measures:

- **Maintainability Effectiveness (MEffec)**: This measure is related to the correctness of the response, and it therefore reflects the ability to maintain the system presented correctly. A higher value of this measure reflects better maintainability effectiveness. It is calculated with the following formula:

\[
MEffec = \frac{\# \text{ correct tasks} - \# \text{ performed tasks}}{\# \text{ tasks}}
\]
• **Maintainability Efficiency (MEffic):** This measure is related to the timing of the response, but also reflects the ability to maintain the system presented correctly. Its unit of measure is “the number of correctly-performed modification tasks per time unit”. The unit of time used was seconds. A higher value of this measure reflects better maintainability efficiency. It is calculated with the following formula:

\[
MEffic = \frac{\# \text{ correct tasks} - \# \text{ performed tasks}}{\text{time spent}}
\]

We also considered a further independent variable (from here on termed as the “cofactor”): **Ability**. We considered this cofactor in our efforts to investigate whether subjects’ abilities play any role in the maintenance of source code, i.e., we discriminate between users according to their respective level of Ability, with the purpose of testing the hypothesis that this is a relevant influencing factor that should be taken into account when adopting these kinds of diagrams. A quantitative assessment of the participants’ Ability was obtained by computing the final grade of the course they were taking. Those students from E-US1 and R-US2 with a final grade of below 5.7/10 (which represents the median of the group) were classified as low Ability participants, while those with a higher grade were given the classification of high Ability students. The instructor of the course (the last author of the paper), who was not one of the experimenters, was asked to provide the grades. In the case of R-UB, the cutoff point was 2.5/5.

4.3.4. **Hypotheses formulation**

The following hypotheses have been formulated and tested:

• **H\(_{1,0}\):** There is no significant difference in the subjects’ maintenance effectiveness when working with UML diagrams which have originated from the design phase or with diagrams which have originated from a Reverse Engineering technique. **H\(_{1,1}\):** \(\neg H_{1,0}\)

• **H\(_{2,0}\):** There is no significant difference in the subjects’ maintenance efficiency when working with UML diagrams which have originated from the design phase or with those which have originated from a Reverse Engineering technique. **H\(_{2,1}\):** \(\neg H_{2,0}\)

The goal of the statistical analysis is to reject the null hypotheses and possibly to accept the alternative ones. Both of the hypotheses are two-sided, because we did not postulate any effect arising from the origin of the diagrams.

4.3.5. **Experimental design**

When designing the experiment we attempted to alleviate several issues that might threaten the validity of the research done by considering the suggestions provided in (Wohlin et al., 2012).

We selected a between-subjects balanced design in which each treatment has an equal number of subjects (Kirk, 1995). We decided to use a between-subjects design
rather than a within-subjects design owing to time constraints. The inherent threats of a between-subjects design were thus alleviated by taking into account the suggestions provided in (Wohlin et al., 2012). In an attempt to alleviate experience effects, we provided the subjects with a background questionnaire (which can be downloaded with the rest of the experiment material) in the training session which took place before carrying out the experiment. The background questionnaire consisted of three blocks of questions. The first one was related to the subjects’ skills and experience. The second block consisted of a set of question used to measure the subjects’ knowledge of UML. The last block was a set of questions about Java. This background questionnaire improved on the design of this family of experiments in comparison with the previous experimentation work presented in the related work section.

The subjects were then assigned to the 2 groups in a random manner (see Table 4.3), based on the marks obtained in the background questionnaire (blocked design by experience). This was not possible in the case of the R-UB subjects, who were assigned to each group in a random manner.

To avoid skewing the results of the tasks as a result of their being of different levels of difficulty, they were randomized. The subjects in each group therefore received the same tasks but in a different order. In order to alleviate learning effects, the order of the tasks was the same for each treatment, i.e., one subject from each group received the tasks in the same order, but in a different order from the rest of his/her group.

Table 4.3. Experimental design.

<table>
<thead>
<tr>
<th>Origin of UML diagrams</th>
<th>RE</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Group 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.6. Experimental tasks

The modification questionnaires were formed of two kinds of maintenance tasks (Table 4.4); both of these activities involve changing the source code:

- **Adaptive maintenance task**: these maintenance activities were intended to enhance the system by adding features, capabilities, and functions, in response to new technology, upgrades, new requirements, or new problems, i.e., a modification of a software product performed after delivery to keep a software product usable in a changed or changing environment (ISO/IEC, 1999). In our case, new requirements had to be added to the system, with the subjects receiving a list of requirements which had to be used to modify the code of the system and thus add/change certain functionalities. This part of the experiment consisted of 3 tasks (see example in Figure 4.2).

- **Corrective maintenance task**: these maintenance activities were “intended to remove errors or bugs from the software, the procedures, the hardware, the network, the data structures, and the documentation” (Swanson, 1976). In our case,
bugs from the source code had to be detected and fixed. We consequently analyzed the list of bugs reported by a professional Dutch IT development company (whose name is not shown for reasons of privacy) and introduced these kinds of defects into our system, giving the subjects a list of functional defects which had to be detected and corrected. All this explains why we consider these tasks to be common, realistic tasks. This part of the experiment consisted of 2 such tasks (see example in Figure 4.3). The subjects were provided with answer sheets for this kind of questions, to allow them to structure their responses.

Figure 4.2. Example of adaptive maintenance task.

Figure 4.3. Example of corrective maintenance task.

These two kinds of tasks needed to be answered using data collection forms, i.e., templates which had to be filled in with pieces of code (see example in Figure 4.4).
We used these data collection forms to obtain a structured response which facilitated the correction of the results, and is also an improvement on the design of this family of experiments in comparison with previous work. The subjects were provided with answer sheets to allow them to structure their responses regarding the maintenance tasks. The reason for doing this was that maintaining source code on paper is not easy owing to space constraints, and the subjects were therefore required to write changes to the source code in a structured manner on the answer sheets (format: line-no, change type, Java code, etc.). They had to fill in a different form depending on the element that they wished to maintain (a class, a method, an attribute, etc.). These answer sheets can be found at: http://alarcos.esi.uclm.es/originUMLmaintenance/

![Figure 4.4. Answer sheet related to the element “Class”, filled with an example.](image)

**Table 4.4. Summary of maintenance tasks.**

<table>
<thead>
<tr>
<th>Task</th>
<th>Summary of task descriptions</th>
<th>Type of maintenance</th>
<th>Maximum mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>When one of the sport center’s services is not available (owing to a breakdown, for example) all reservations for this service should be cancelled.</td>
<td>Corrective</td>
<td>4 points</td>
</tr>
<tr>
<td>T2</td>
<td>The sport center’s system should store its customers’ telephone numbers.</td>
<td>Adaptive</td>
<td>5 points</td>
</tr>
<tr>
<td>T3</td>
<td>A ticket showing a customer’s reservations at a specific time should be generated by the system.</td>
<td>Adaptive</td>
<td>5 points</td>
</tr>
<tr>
<td>T4</td>
<td>When we delete one of the sport center’s members, his/her pending payments sometimes remain in the system.</td>
<td>Corrective</td>
<td>2 points</td>
</tr>
<tr>
<td>T5</td>
<td>The information about the sport center’s instructors should be stored by the system.</td>
<td>Adaptive</td>
<td>6 points</td>
</tr>
</tbody>
</table>
The largest change consisted of adding a class which would need at least 22 lines of code. In general, between 1 and 3 classes needed to be modified. The complexity of the task might not appear to be too complex, owing to the number of LoCs that have to be changed, but the complexity of the task lies in the difficulty of detecting where the change is to be made in the source code, and how it should be carried out. It should also be borne in mind that 5 tasks had to be completed in 2 hours, using a system that the subjects had never seen. We limited the time of the experiment to fit in with subjects’ availability. The subjects were only required to maintain the system, i.e., they did not need to update diagrams according to their changes or to create test cases.

After performing each maintenance task, the subjects were also required to indicate which artifacts (source code, class diagrams and/or sequence diagrams) they had used to solve the task. They were asked this in order to check whether or not they used the diagrams to solve the maintenance tasks (otherwise, the effect measured would not be the influence of the different origin of the diagrams). We shall refer to these kinds of questions as “post-questions”.

Table 4.5. Post-Experiment Survey.

<table>
<thead>
<tr>
<th>Id</th>
<th>Question/Issue</th>
<th>Possible Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex1</td>
<td>The difficulty of tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex2</td>
<td>The training was sufficient to be able to perform the tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex3</td>
<td>The clarity of the material provided</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex4</td>
<td>The task objectives were perfectly clear to me</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex5</td>
<td>The tasks I performed were perfectly clear to me</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex6</td>
<td>I did not experience difficulty in reading the diagrams</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex7</td>
<td>I did not experience difficulty in reading the source code</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex8</td>
<td>The LoD of the diagrams was correct enough for me to be able to perform the tasks</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex9</td>
<td>The available class diagrams were helpful</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex10</td>
<td>In the event that you do not think that the class diagrams have been useful, indicate why</td>
<td>Open question</td>
</tr>
<tr>
<td>Ex11</td>
<td>The available sequence diagrams were helpful</td>
<td>(1-5)</td>
</tr>
<tr>
<td>Ex12</td>
<td>In the event that you do not think that the sequence diagrams have been useful, indicate why</td>
<td>Open question</td>
</tr>
<tr>
<td>Ex13</td>
<td>I had enough time to perform the tasks</td>
<td>Multiple choice question</td>
</tr>
<tr>
<td>Ex14</td>
<td>How much time (as a percentage) did you spend looking at the diagrams?</td>
<td>Multiple choice question</td>
</tr>
<tr>
<td>Ex15</td>
<td>How much time (as a percentage) did you spend looking at the source code?</td>
<td>Multiple choice question</td>
</tr>
</tbody>
</table>

1 = strongly agree; 2 = agree; 3 = neutral; 4 = disagree; 5 = strongly disagree (Ex2, Ex3, Ex4, Ex5, Ex6, Ex7, Ex9, Ex11)

1 = very high; 2 = high; 3 = correct; 4 = low; 5 = very low (Ex8)

1 = very difficult; 2 = difficult; 3 = medium; 4 = easy; 5 = very easy (Ex1)

1 = very clear; 2 = clear; 3 = correct; 4 = unclear; 5 = very unclear (Ex3)

A = more time needed; B = less time needed; C = enough time (Ex13)

A. <20%; B. >=20% and <40%; C. >=40% and <60%; D. >=60% and <80%; E. >=80% (Ex14, Ex15)
In addition, at the end of the experiment’s execution the subjects were asked to fill in a post-experiment survey (see Table 4.5), whose goal was to obtain feedback about their perception of the experiment execution - feedback which could be used to explain the results obtained. The answers to the questions were based on a five-point Likert scale (Oppenheim, 2000). During the experiment’s execution, the subjects had to perform 5 maintenance tasks, in different orders, which are summarized in Table 4.3.

### 4.3.7. Experimental procedure

The experimental material and the time duration were checked by carrying out a pilot study with 6 PhD students from the University of Castilla-La Mancha, Spain, before the execution of the first experiment. The pilot study was similar to the experiment described in this section, but with no time limit. The results of the pilot study were used as a basis to adapt the number of tasks and their complexity to the experimental time constraints. Some spelling mistakes were also corrected and some requirement statements were rewritten in order to make them more understandable.

We performed internal replications of the original experiment, and the entire family of experiments was carried out by the same experimenters. We conducted the experiment and the replications in a classroom under controlled conditions.

We did not provide details on the experimental hypotheses, and informed the participants that their grade on the course would not be affected by their performance.

The experiment and replications took place in two sessions of two hours each:

- **Training session:** The subjects first attended a training session in which detailed instructions on the experiment were presented and the main concepts of UML and JAVA were revised. In this session, we did not provide details on the experimental hypotheses and the subjects carried out an exercise similar to those in the experimental tasks in collaboration with the instructor. During the training session, the subjects were required to fill in a background questionnaire. The participants were informed that the data collected in the experiments were to be used for research purposes and would be treated as confidential, and that their grade on the course they were taking would not be affected by the grade obtained in the experiment. After the introductory lesson, we assigned the participants to one of the 2 groups in accordance with the marks obtained in the background questionnaire, thus obtaining balanced groups. Those subjects who did not come to the training session were randomly assigned to the groups, which in some cases led to unbalanced groups (a difference of 1 subject).

- **Execution session:** The execution of the experiment took place in the second session, in a classroom, where the students were supervised by the instructor of the course (a different one depending on the replication) and one experimenter (always the same one), and no communication between them was allowed. Each of the groups was given a different treatment.
After the execution of the experiment, the data collected from it were placed on an excel sheet, following an answering diagram constructed before the experiment was carried out. On this sheet, each task has a maximum mark (see Table 4.4), depending on the correctness of the answer provided. This means that for each task, a mark was given to the subject depending on the number of correct lines of code added to the solution. Incorrect answers were not given negative marks, i.e., lines of code which do not solve the task.

4.3.8. Analysis procedure

The data analysis was carried out by considering the following steps:

1. We first carried out a descriptive study of the measures of the dependent variable, i.e., MEffec and MEffic in order to obtain a general overview of the influence of the main factor (Origin).

2. We then analyzed the characteristics of the data to determine which parametric or non-parametric test it would be better to use. We performed a Kolmogorov-Smirnov test (Sheskin, 2007) to determine the normality of distributions and a Levene (Sheskin, 2007) test to determine the homogeneity of variances. These analyses are useful in determining which parametric or non-parametric test it is best to use.

3. Based on the results of the previous test, for the data collected in each experiment we tested the hypotheses formulated using the parametric ANOVA test (Devore and Farnum, 1999) when the results of Kolmogorov-Smirnov test and Levene test were positive. The non-parametric Mann-Whitney test (Sheskin, 2007) was performed when the data obtained did not satisfy the restrictions of the ANOVA test (we did not obtain normal distributions, and there is no homogeneity of variances).

4. To strengthen the results of each experiment, we decided to integrate them using a meta-analysis. A meta-analysis is a set of statistical techniques which are used to combine the different effect sizes of the experiments in order to obtain a global effect of a factor in a dependent variable.

5. We analyzed the influence and the interaction of the cofactor, i.e., Ability, using statistical tests. The interaction of the Ability with the main factor (i.e., Origin) was also tested using interaction plots (Devore and Farnum, 1999). Interaction plots are simple line graphs in which the means on the values of a dependent variable for each level of one factor are plotted on all the levels of the second factor. The resulting lines are parallel when there is no interaction and nonparallel when an interaction is present.

6. The data collected from the post-experiment survey were eventually analyzed using bar graphs. In cases in which a pattern was detected in the data, these data were also tested with a T-test (Sheskin, 2007) owing to their nature.

All the statistical values were calculated using SPSS (SPSS, 2003), with its standard configuration. In all the statistical tests, we decided to accept a probability of 5% of committing a Type-I-Error (Sheskin, 2007).
4.3.9. Documentation and communication

Issues such as documentation (Shull et al., 2004) and communication among experimenters (Vegas et al., 2006) may influence the success or the failure of an experiment performance and its future replications. Laboratory packages and knowledge-sharing mechanisms were used to handle these issues. The material was originally written in Spanish for E-US1 and R-US2, and was then translated into Italian for R-UB. The material included: the post-experiment survey, the modification questionnaires, the data collection forms, the source code and the UML diagrams (two versions: D and RE). The groups of experimenters also shared a document to provide a common background so as to be able to communicate all terms related to the design and analysis of the experiment.

The experimenters (the first three authors of the paper) began with an initial face-to-face meeting in which the main ideas of the experiments were discussed and reported in an agreement document. All the experimenters then exchanged the agreement documents from the meeting by e-mail to attain a shared common research plan. This phase played a significant role in sharing knowledge among the experimenters and in the discussions on possible issues related to the study that might arise.

The experimenters used instant messaging tools and e-mails to establish a communication channel in all phases of the study. Teleconferences were also held to share knowledge among the research groups and to discuss the experimental procedure that the participants had to follow.

4.4. RESULTS

In this section, we present the data analysis following the procedure presented above: the presentation of the descriptive statistics, the test of the hypotheses related to the main factor (Origin), the analysis of the influence of the cofactor Ability and the analysis of the post-experiment survey.

Please bear in mind that in some cases the groups are not balanced (there are more subjects in one group than in the other) owing to the fact that some subjects abandoned the experiment during its execution.

4.4.1. Descriptive statistics and exploratory analysis

Table 4.6 and Table 4.7 show the descriptive statistics of the Maintainability measures, i.e, MEffec and MEffic respectively, grouped by the Origin of the UML diagrams. These tables contain the following data: number of subjects (N), mean ($\bar{X}$), median, and standard deviation (SD).

At a glance, we can observe that when the subjects used FD diagrams they obtained better values (although the difference is very low in both measures when comparing means (only in the case of MEffec in R-UB are the results are better with RE diagrams). This indicates that there is a slight tendency towards FD diagrams improving the performance of software code maintainers.
Table 4.6. Descriptive statistics for MEffec.

<table>
<thead>
<tr>
<th>Origin</th>
<th>N</th>
<th>(\bar{X})</th>
<th>Median</th>
<th>SD</th>
<th>N</th>
<th>(\bar{X})</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-US1</td>
<td>20</td>
<td>0.641</td>
<td>0.6818</td>
<td>0.165</td>
<td>20</td>
<td>0.650</td>
<td>0.6818</td>
<td>0.148</td>
</tr>
<tr>
<td>R-US2</td>
<td>23</td>
<td>0.597</td>
<td>0.5909</td>
<td>0.223</td>
<td>28</td>
<td>0.667</td>
<td>0.6818</td>
<td>0.174</td>
</tr>
<tr>
<td>R-UB</td>
<td>39</td>
<td>0.512</td>
<td>0.5000</td>
<td>0.171</td>
<td>39</td>
<td>0.441</td>
<td>0.4091</td>
<td>0.219</td>
</tr>
</tbody>
</table>

Table 4.7. Descriptive statistics for MEffic.

<table>
<thead>
<tr>
<th>Origin</th>
<th>N</th>
<th>(\bar{X})</th>
<th>Median</th>
<th>SD</th>
<th>N</th>
<th>(\bar{X})</th>
<th>Median</th>
<th>SD</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-US1</td>
<td>20</td>
<td>0.00270</td>
<td>0.00283</td>
<td>0.00079</td>
<td>20</td>
<td>0.00273</td>
<td>0.00303</td>
<td>0.00072</td>
</tr>
<tr>
<td>R-US2</td>
<td>22</td>
<td>0.00261</td>
<td>0.00245</td>
<td>0.00106</td>
<td>28</td>
<td>0.00350</td>
<td>0.00352</td>
<td>0.00147</td>
</tr>
<tr>
<td>R-UB</td>
<td>39</td>
<td>0.00217</td>
<td>0.00204</td>
<td>0.00087</td>
<td>39</td>
<td>0.00255</td>
<td>0.00183</td>
<td>0.00406</td>
</tr>
</tbody>
</table>

4.4.2. Influence of Origin of diagram

In order to test the formulated hypotheses \(H_{1.0}, H_{2.0}\) we analyzed the effect of the main factor (i.e. \textit{Origin}) on the measures considered (i.e., MEffec and MEffic) using the non-parametric Mann-Whitney or ANOVA test, depending on the normality of data (as explained in Section 4.3.7).

In the following subsections, the results for each measure of the Mann-Whitney U tests or ANOVA tests are shown in tables (Table 4.8, and Table 4.9), in which the \textit{Origin} column describes the independent variable, \textit{p}-value is the statistical significance obtained, \textit{op} is the estimated observed power of the test, \textit{es} is the effect size, and \textit{R} describes whether the data obtained allows us to reject the null hypothesis, while the tendency of the data in the case of the null hypothesis being rejected is shown in the “\textit{in favour of}…” column.

The results obtained for each hypothesis will be commented on in their corresponding subsections.

For each measure, we first decided to analyze the data related to maintenance in general, as is presented in the formulated hypothesis. We then made the decision to analyze the results by dividing them by the type of maintenance, since there may have been differences between the results from the adaptive and the corrective maintenance.

4.4.2.1. Testing Maintenance Effectiveness: MEffec (\(H_{1.0}\))

The first row of Table 4.8 shows that we cannot reject \(H_{1.0}\) in the first experiment (E-US1) and the first replication (R-US2) given that their \textit{p}-value is greater than 0.05. Hence, here the different origins of UML diagrams had no effect on the subjects’ effectiveness when performing the source code maintenance tasks. The observed power of the test is low, probably because of a small effect size, so we assume a 0.946 (1 - 0.054) and a 0.755 (1 – 0.245) estimated probability of a Type II error in our
assertions. Given the low value of the observed power, we cannot obtain strong conclusions from E-US1 and R-US2.

In the case of the second replication, the p-value is 0.047, i.e., lower than 0.05, and we can therefore reject the null hypothesis. In this case we can assume that there is a difference in the maintenance effectiveness when using RE diagrams or FD diagrams in favor of RE diagrams. What is more, the power of the test is quite high in this case but it is still not sufficient for us to be confident of the results, because there is a probability of 65% of committing a Type II error in our assertion.

Table 4.8. Statistical relation between Origin of diagram (RE/FD) and Maintainability Effectiveness (MEffec).

<table>
<thead>
<tr>
<th>Origin</th>
<th>Test</th>
<th>p-value</th>
<th>op</th>
<th>es</th>
<th>R</th>
<th>In favour of…</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-US1</td>
<td>Man-Withney</td>
<td>0.957</td>
<td>0.054</td>
<td>0.001</td>
<td>NO</td>
<td>-</td>
</tr>
<tr>
<td>R-US2</td>
<td>ANOVA</td>
<td>0.202</td>
<td>0.245</td>
<td>0.033</td>
<td>NO</td>
<td>-</td>
</tr>
<tr>
<td>R-UB</td>
<td>Man-Withney</td>
<td>0.047</td>
<td>0.352</td>
<td>0.033</td>
<td>YES</td>
<td>RE diagrams</td>
</tr>
</tbody>
</table>

We also performed an analysis of the influence of the Origin on maintenance effectiveness per type of maintenance, i.e., adaptive and corrective maintenance. The results were not significant in all the cases because the p-value is always greater than 0.05. The same occurred when the MEffec as regards the time spent maintaining the system (without relating this to the number of correct answers) was tested.

4.4.2.2. Testing Maintenance Efficiency: MEffic (H₂,₀)

The numbers in Table 4.9 show that in the case of E-US1 and R-UB there are no significant effects (the p-values are not smaller than 0.05) as regards the Origin of UML diagrams on maintenance efficiency and that, in this case, the statistical power is still very low. But, if the null hypothesis were to be accepted, we would be assuming a 0.949 (i.e., 1-0.051) estimated probability of a Type II error.

But as was the case with MEffec, there is one case in which there is an influence of the Origin on the maintainability efficiency in favor of FD diagrams. In this case, in R-US2 the p-value is 0.049, and the null hypothesis is therefore rejected.

Table 4.9. Statistical relation between Origin of diagram (RE/FD) and Maintainability Efficiency (MEffec).

<table>
<thead>
<tr>
<th>Origin</th>
<th>Test</th>
<th>p-value</th>
<th>op</th>
<th>es</th>
<th>R</th>
<th>In favour of…</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-US1</td>
<td>Man-Withney</td>
<td>0.534</td>
<td>0.051</td>
<td>0.0003</td>
<td>NO</td>
<td>-</td>
</tr>
<tr>
<td>R-US2</td>
<td>Man-Withney</td>
<td>0.049</td>
<td>0.343</td>
<td>0.049</td>
<td>YES</td>
<td>FD diagrams</td>
</tr>
<tr>
<td>R-UB</td>
<td>Man-Withney</td>
<td>0.272</td>
<td>0.088</td>
<td>0.004</td>
<td>NO</td>
<td>-</td>
</tr>
</tbody>
</table>

Once again, an analysis of the influence of the Origin on maintenance efficiency per type of maintenance, i.e., adaptive and corrective maintenance, was performed. In this case, the results were not significant either (the p-values are greater than 0.05).

We also attempted to measure MEffic as regards the time spent maintaining the system, without relating this to the number of correct answers (as was done before). In
this case, the p-value was again higher than 0.05 (i.e., p-value=0.725) but with a higher statistical power (i.e., op=0.5). The same result was obtained in the test with the MEffic measure as regards the time spent maintaining the system (without relating this to the number of correct answers).

4.4.2.3. Integrating the obtained results through meta-analysis

When the different effect sizes of the experiments need to be combined to obtain a global effect of a factor, the statistical technique used is that of meta-analysis. Although we have found some significant results: in favor of RE diagrams in relation to MEffic in R-UB, and in favor of FD diagrams in relation to MEffic in R-US2, there are no significant results in the remaining cases. We have therefore decided to integrate the results of the different studies through a meta-analysis (in which the factor is the Origin of UML diagrams and how this affects the modifiability of source code), in order to explore if stronger results can be found. This technique has been used for the same purpose in other families of experiments, such as that shown in (Cruz-Lemus et al., 2010).

For each dependent variable, we computed the mean value obtained by the participants when using RE diagrams, minus the mean value they obtained with FD diagrams, and these values were then used to compute the Hedges’ g metric (Hedges and Olkin, 1985; Kampenes et al., 2007). The overall conclusion was obtained by calculating the Z score based on the mean and standard deviation of the Hedges’ g statistics of the experiments. The global effect size was therefore obtained by using the Hedges’ g metric, with the weights proportional to the experiment size:

$$Z = \frac{\sum w_i z_i}{\sum w_i}$$

where $w_i = 1/(n_i-3)$ and $n_i$ is the sample size of the $i$-th experiment. The higher the value of Hedges’ g, the higher the corresponding mean difference. An effect size of 0.5 indicates that the mean value obtained when using FD diagrams is half a standard deviation larger than the mean value obtained when not using them.

As suggested in (Kampenes et al., 2007), the effect size can be classified as: small (S) for values between 0 and 0.37, medium (M) for values between 0.38 and 1.0, and large (L) for values above 1.00.

The meta-analysis was performed by using the Comprehensive Meta-Analysis v2 tool (Biostat Inc., 2006). For each measure, the tool produced the forest plots depicted in Figure 4.5 and Figure 4.6. The squares and diamonds are mostly proportional in size to each study’s weight under the fixed effect model (see the ‘Relative weight’ column). The squares show the individual effect size of each experiment and the diamond shows the global effect size. The values of the Hedges’ g metric are also reported. Positive values indicate that the use of FD diagrams improves the modifiability of source code. Negative values signify that RE diagrams are the best treatment.
Chapter 4: Are Forward Designed or Reverse-Engineered UML Diagrams More Helpful for Code Maintenance?

If we focus on the results obtained for the MEffec variable (see Figure 4.5), the total effect is in favor of using FD diagrams, but the global effect size obtained is not statistically significant since the p-value is greater than 0.05. The value obtained for the Hedge’s g metric, i.e., 0.282, indicates a small size for the global effect. Similar results were obtained for MEffic (see Figure 4.6), where the total effect is again in favor of FD diagrams, but the size of the effect is small. In both cases, if the coincidence interval were to be increased to 90% instead of 95%, then the meta-analysis would be significant because the p-values would not be greater than 0.10.

![Figure 4.5. Meta-analysis results for Maintainability Effectiveness (MEffec).](image)

![Figure 4.6. Meta-analysis results for Maintainability Efficiency (MEffic).](image)
4.4.3. **Influence of Ability**

We tested whether the subjects’ *Ability* influenced the results but. The effect of this factor could not be confirmed because the p-value found was higher than 0.05 in all of the cases (see Table 4.10), i.e., the subjects’ *Ability* had no statistical influence on the results. This was expected owing to the balanced design of the experiment.

The interaction plot shown in Figure 4.7 indicates that there was no interaction between *Origin* and *Ability* for MEffec. In the case of E-US1 and R-US2, high ability participants achieved better scores than low ability ones when both of them were using RE and FD diagrams. The interaction plot also suggests that the results achieved with FD diagrams are better than those obtained with RE diagrams for both high and low ability participants. This might be owing to the fact that RE diagrams contain too many details when compared with FD diagrams. In particular, RE sequence diagrams are twice as large in terms of messages when compared to FD diagrams. This could be because FD diagrams only contain logical messages between objects, and that messages between other kinds of objects are obviated, such as objects from Java packages, which are shown in RE diagrams. This difference between RE and FD diagrams is based on their nature, owing to the fact that human based diagrams contain less technical details than RE diagrams. In the case of R-UB, the effect of the combination of the Ability and the Origin is contrary to the other cases, i.e., the results achieved with RE diagrams are better than those obtained with FD diagrams for both high and low ability participants.

Table 4.10. Statistical relation between Ability and Maintainability Effectiveness (MEffec) and Efficiency (MEffic).

<table>
<thead>
<tr>
<th>Ability</th>
<th>MEffec</th>
<th>MEffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-US1</td>
<td>0.226</td>
<td>0.914</td>
</tr>
<tr>
<td>R-US2</td>
<td>0.460</td>
<td>0.939</td>
</tr>
<tr>
<td>R-UB</td>
<td>0.470</td>
<td>0.995</td>
</tr>
</tbody>
</table>

Figure 4.7. Interaction between Origin and Ability for Maintainability Effectiveness (MEffec).

As with MEffec, the influence of the subjects’ *Ability* and the *Origin* in the case of MEffic was also represented on an interaction plot. The interaction plot shown in
Figure 4.8 indicates that there was a clear interaction between \textit{Origin} and \textit{Ability} for MEffic. In this case, high \textit{Ability} participants achieved better scores using the FD diagrams, and low \textit{Ability} participants did better using the RE diagrams (except in the case of R-US2). This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams. However, experienced maintainers do not need very high traceability, because using FD diagrams might allow them to obtain sufficient information to attain a correct overview of how the system works.

4.4.4. \textbf{Post-experiment survey results}

The analysis of the answers to the post-experiment survey revealed that the time needed to carry out the modification tasks (Figure 4.9) was not considered to be sufficient (more time was needed), and that the subjects considered that the performance of the tasks was of more or less low-medium difficulty (Figure 4.10), independently of the particular treatment received. The need for more time to perform the tasks may have arisen from the fact that the measurement of the time needed was derived from the pilot study, which was performed by PhD students, who probably had higher ability and/or more experience than these Master’s students, signifying that the less experienced subjects needed more time. We would also like to state that some subjects did not finish the questionnaire owing precisely to this lack of time.

We also asked about the subjects’ perception of the adequacy of the LoD of the diagrams used. The majority of the subjects who received FD diagrams agreed with the LoD of the diagrams they received (i.e., they considered the LoD as the “right” amount of detail). In the case of those subjects who received RE diagrams, the majority agreed with the LoD presented, but of those subjects who did not, the majority required more LoD (Figure 4.11).
Figure 4.9. Subjects’ answers as regards adequacy of time provided.

Figure 4.10. Subjects’ answers as regards difficulty of task.

Figure 4.11. Subjects’ answers as regards adequacy of the LoD.
The subjects who received FD diagrams experienced fewer difficulties when reading the diagrams used, in comparison with the RE group, as is shown in Figure 4.12. We tested whether there was a difference as regards the difficulties experienced by subjects depending on the diagrams they used. This was done by using a Mann-Whitney test owing to the nature of the data, and this test was carried out by comparing the subjects’ responses (from 1=very low to 5=very high) grouped by the UML diagrams that they had used (RE or FD diagrams). The results of the test show a significant difference (p-value=0.007, which is lower than α=0.05). The power of the test is high (0.743), and this therefore allows us to state that the subjects who received RE diagrams experienced more difficulties when reading diagrams than those who received FD diagrams.

Figure 4.12. Subjects’ answers as regards to difficulties when reading diagrams.

Figure 4.13. Subjects’ answers as regards usefulness of class diagram used.
As part of the post-experiment survey, the subjects were required to indicate how useful the diagrams were for them in general as regards solving tasks. Class diagrams are considered useful in both groups, in more or less the same proportion (Figure 4.13). Having said that, the majority of the subjects who received the RE diagrams commented that the sequence diagrams employed were not useful and were very difficult to understand, as opposed to the majority of the subjects in the D group who considered them helpful (Figure 4.14). This finding may have been caused by the different complexities and varying LoD in the different kinds of diagrams, as explained in previous sections.

A Mann-Whitney test was then used (owing to the nature of the data) to test whether there was a difference as regards the subjects’ perceived usefulness of the sequence diagrams depending on the diagrams they used. The test was carried out with the same configuration as explained in the test for the difficulties when reading the diagrams. The results of the test again show a significant difference (p-value=0.002, which is lower than α=0.05). The power of the test is high (0.88), and this therefore allows us to state that the subjects who received RE sequence diagrams considered them to be less helpful than the sequence diagrams in comparison to those who received FD sequence diagrams.

![Figure 4.14. Subjects’ answers as regards usefulness of sequence diagrams used.](image)

The results of the post-questions concerning the artifacts used to answer the questions led us to detect that almost all the subjects used the source code to solve the tasks. This was expected, in the sense that source code is needed when it is being maintained. The use of the source code was from 7 to 25% higher in the RE group than in the D group.

We then analyzed whether or not the subjects had used the diagrams (Table 4.11). Class diagrams were also used by the majority of subjects (except in the case of the D group in R-UB). These percentages are consistent with the subjective response
provided in the post-experiment survey (see Figure 4.13). In the case of the RE group, the subjects used class diagrams in more or less the same proportion for corrective or perfective tasks, but in the case of the D group, the subjects used from 7 to 27% more class diagrams for perfective tasks. This may have occurred because class diagrams provide the structure of the system, thus allowing maintainers to obtain an overview of the system faster, which would appear to be easier with the FD diagrams owing to their conciseness; this is more important for perfective tasks. If we focus on the use of sequence diagrams, we would like to highlight that their use was surprisingly low; in general, only 20 to 41% of the subjects used them. That is consistent with the RE group subjects’ opinions (Figure 4.14), in which they indicate that they did not use sequence diagrams, and they also believe that they are not useful diagrams for understanding the system during its maintenance. In the case of the D group, there is an inconsistency arising from the fact that the subjects did not use sequence diagrams in most of the tasks, even though they considered them to be useful (see Figure 4.14). Subjects from both groups (except the RE group of R-US2) used the sequence diagram more in corrective tasks than in perfective tasks (a difference of 2 to 50%). The reason for this could be that for corrective tasks, in which maintainers need to localize an error, structure and behavior are needed, since the error might be caused by a structural error or by a behavior error.

<table>
<thead>
<tr>
<th>Exp</th>
<th>Usage of class diagrams</th>
<th>Usage of sequence diagrams</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RE</td>
<td>D</td>
</tr>
<tr>
<td>E-US1</td>
<td>80%</td>
<td>74%</td>
</tr>
<tr>
<td>R-US2</td>
<td>59%</td>
<td>69%</td>
</tr>
<tr>
<td>R-UB</td>
<td>67%</td>
<td>40%</td>
</tr>
</tbody>
</table>

4.4.5. Summary and discussion of the data analysis

The experiment (E-US1) was performed with 40 students from the University of Seville (Spain), and it was replicated in the same university with 51 students (R-US2) and in the University of Bari, Italy, with 78 students (R-UB).

Descriptive statistic results show that subjects using FD UML diagrams obtained better values in both measures (except in the case of MEffic in R-UB), indicating that FD diagrams may, to some extent, improve the maintenance of the source code, but that the differences are very slight.

As regards the results of the statistical test, in almost all of the cases the variables (i.e., MEffec and MEffic) are not significantly affected by the Origin of the UML diagrams, i.e., the results of the tests performed did not allow us to reject any of the null hypotheses presented in section 4.3 in almost all cases, as all the significance levels are above 0.05. The test powers are low, so the possibility of an error occurring as a result of accepting the null hypothesis is high. But if we focus on the case of R-US2, the results of the MEffic were influenced by the Origin, thus obtaining a high
test power. The same occurred in the case of R-UB with MEffec, where there is a clear difference in favor of FD diagrams.

Despite the fact that the results were not conclusive in all the cases (which is not as positive as we expected), we have ensured that the experimental results were not influenced by other cofactors such as the subjects’ Ability. If we focus on the interaction between Origin and Ability, in the case of E-US1 and R-US2, the subjects using FD diagrams achieved better results than those who used RE diagrams, in the case of both high and low ability participants. This may have been caused by the fact that RE diagrams contain too many details when compared with FD diagrams. In the case of R-UB, the effect of the combination of the Ability and the Origin is contrary to the other cases, i.e., the results achieved with RE diagrams are better than those obtained with FD diagrams in the case of both high and low ability participants. The low ability users obtained more benefits from RE diagrams than from FD diagrams in terms of efficiency owing to the high traceability between RE diagrams and code.

The results of the family of the experiments are summarized in Table 4.12.

<table>
<thead>
<tr>
<th>Exp_ID</th>
<th>Descriptive statistics (in favour of...)</th>
<th>Influence of Origin</th>
<th>Influence of Ability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MEffec</td>
<td>MEffic</td>
<td>MEffec</td>
</tr>
<tr>
<td>E-US1</td>
<td>FD</td>
<td>FD</td>
<td>×</td>
</tr>
<tr>
<td>R-US2</td>
<td>FD</td>
<td>FD</td>
<td>×</td>
</tr>
<tr>
<td>R-UB</td>
<td>RE</td>
<td>FD</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Legend**

- Descriptive statistics:
  - “FD” = better results when using FD diagrams than RE diagrams
  - “RE” = better results when using RE diagrams than FD diagrams
  - “-” = no differences when using FD or RE diagrams

- Influence of Origin:
  - ×  = hypothesis not rejected → There is no significant difference in the results when working with FD or RE UML diagrams
  - ✓  = hypothesis rejected → There is a significant difference in the results when working with FD or RE UML diagrams (expected value)

- Influence of Ability:
  - ×  = hypothesis not rejected → There is no significant difference in the results when working with high or low ability students (expected value)
  - ✓  = hypothesis rejected → There is a significant difference in the results when working with high or low ability students

After the individual analysis, a meta-analysis was performed in order to clarify the results. Its results also show a tendency in favor of FD diagrams for both measures (MEffec and MEffic). But in both cases, the results are not clearly evident owing to the values of the p-values.
Moreover, if we study the results of the post-experiment survey, we can see better subjective results for the FD diagrams. This is because the subjects who received RE diagrams did not believe their sequence diagrams to be useful, since they were not understandable. Significant results were obtained, showing that the subjects who received RE diagrams experienced more difficulties when reading the diagrams used; this is especially true as regards sequence diagrams. In the case of the D group, there was an inconsistency arising from the fact that the subjects did not use sequence diagrams in most of the tasks, even though they considered them to be useful.

We would like to underline that, according to the results of the post-question for each maintenance task concerning the use of artifacts in task solving, UML class diagrams were used in more or less in the same proportion as the source code for system understanding during the maintenance tasks. The sequence diagram was less widely-used, probably because of the nature of the tasks presented during the course of this experiment (more perfective maintenance tasks were required than corrective maintenance tasks). As stated previously, UML diagrams are not usually updated during maintenance tasks owing to the time constraints in realistic environments. But the high level of use of class diagrams during this experiment leads us to recommend companies to keep them up to date in order to improve maintainers’ performances.

4.5. IMPLICATIONS OF THE STUDY

No evident overall pattern emerges from the family of experiments of this study – especially when limiting to the statistics of the experiments: when considered individually 1 out of 3 experiments does not provide statistically significant results in favour of either RE or FD diagrams.

The experiments showed that high Ability participants achieved better scores using the FD diagrams, and low Ability participants did better using the RE diagrams (except in the case of R-US2). This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams. This group may find it difficult to make mentally bridge the gap between source code and high level class diagram. However, experienced maintainers do not need very high traceability, because using FD diagrams might allow them to obtain sufficient information to attain a correct overview of how the system works.

The key characteristics of RE and FD diagrams that we compare in this study are the following: RE diagrams are complete and close to source code. Man-made FD diagrams are almost never complete, but their strength seems to be the selective inclusion of information about the system and modest complexity of the diagrams.

Findings from the post-experiment survey show that subjects who received FD diagrams experienced fewer difficulties when reading the diagrams compared to the RE group (p-value=0.007), especially with the class diagrams. At the same time, RE sequence diagrams were found less useful than FD sequence diagrams (p-value=0.002). A possible explanation for this is that human engineers make (implicit)
assessments regarding which information is important for understanding key aspects of the system and thus regarding which information should be included in a diagram. We believe the findings of our experiments indicate that FD class diagrams provide a more attractive balance between detail and relevant information than RE class diagrams. For sequence diagrams, it seems that current automated reverse engineering methods are not able to filter out the relevant information for maintenance tasks.

On the other hand, for performing the maintenance tasks in our study, participants found that source code source to be the most useful source of information, closely followed by the class diagrams. The participants therefore perceived that UML diagrams might add little value over access to the source code for the tasks in our study. This result is relevant for the researcher because it might be interesting to investigate the motivation guiding a software engineer when trusting in a source of information and how he/she exploits it to accomplish a maintenance task.

4.6. THREATS TO VALIDITY

We must consider certain issues which may have threatened the validity of the experiment (Wohlin et al., 2012):

- **External validity**: External validity can be threatened when experiments are performed with students, and the representativeness of these subjects may be doubtful in comparison to that of software professionals. In spite of this, the tasks to be performed did not require high levels of industrial experience, so we believe that this experiment could be considered appropriate, as it follows suggestions made in the relevant literature (Basili and Weiss, 1984; Basili et al., 1999; Höst et al., 2000). Working with students also implies various advantages, such as the fact that their prior knowledge is fairly homogeneous, there is the possible availability of a large number of subjects (Verelst, 2004), and there is the opportunity to test experimental design and initial hypotheses (Sjøberg et al., 2005). An additional advantage of using novices as subjects in experiments on maintainability is that the cognitive complexity of the objects under study is not hidden by the subjects’ experience. Nevertheless, it would be extremely interesting to carry out further replications of the experiment with practitioners.

- Another threat to external validity concerns the experimental material used. The selected system is representative of a small industrial system. In addition, the experimental objects are small. The results of the experiments might be different when using bigger systems and experimental objects. The size of the experimental objects could also threaten the external validity of the results. The rationale for selecting the experimental objects used relies on the need (owing to time constraints) to simulate actual maintenance tasks related to small maintenance operations that novice software engineers and/or junior programmers may perform in a software company.

- **Internal validity**: Threats to internal validity were mitigated by the design of the experiment. Each subject was grouped by his/her results in the background
questionnaire, so both groups had subjects with a similar skill level. In all the
cases, the subjects’ knowledge of UML and JAVA was reinforced by teaching
them about UML diagrams and JAVA in a training session organized to take place
the day before the experiment was carried out, but their knowledge was sufficient
for them to understand the given system, and they all had roughly the same
background (which was tested with a background questionnaire).

- Furthermore, all the participants found the material provided, the tasks, and the
goals of the experiment to be clear, as the post-experiment survey results showed.
Another safeguard was that the instrumentation was tested in a pilot study, to check
its validity. In addition, mortality threats were mitigated by offering the subjects
the possibility of performing similar tasks in the final exam of the course that they
were taking. Another issue that is a potential threat is the exchange of information
among the participants. We must emphasize that the participants were not allowed
to communicate with each other, and were prevented from doing so by being
monitored during the run of the experiment. When the experiment had concluded,
the participants were asked to give back all the experimental material.

- **Construct validity**: This validity may be influenced by the measures used to
obtain a quantitative evaluation of the subjects’ performance, the maintenance
tasks, and the post-experiment survey, in addition to social threats. We performed
the experiment in a really short period of time owing to the subjects’ constraints.
The small amount of time that the students were given to perform the tasks could
have influenced the results of this experiment, as could the small number of tasks,
which was again owing to constraints on our subjects’ time. The measures used
were selected to achieve a balance between the correctness and completeness of the
answers, which are well-known measures and are widely-used in this kind of
experiments. The questionnaires were defined to obtain sufficiently complex
questions, without them being too obvious. The post-experiment survey was
designed using standard forms and scales. Social threats (e.g., evaluation
apprehension) have been avoided, since the students were not graded on the results
obtained. Absenteeism was avoided by performing similar tasks to the exercises
that would appear in their final exam.

- Based on the results of these pilot study, we consider that the two experimental
objects fit the time constraints of the experiment in that they are sufficiently
realistic for small maintenance operations that novice software engineers perform
within a software company (Cohen et al., 2004).

- The experimental material delivered to the subjects consisted of the UML diagrams
(class and sequence diagrams) and the JAVA source code of the system, the answer
sheets, and the post-experiment survey.

- **Conclusion validity**: Conclusion validity concerns the data collection, the
reliability of the measurement, and the validity of the statistical tests, all or any of
which might affect the ability to draw a correct conclusion. Statistical tests were
used to reject the null hypotheses, but the fact that subjects performed a small
number of tasks provided us with few data points to work with. These particular
statistical tests were selected by checking that they followed the specific assumptions related to their use. We have explicitly mentioned and discussed all those cases in which non-significant differences were present.

4.7. CONCLUSIONS AND FUTURE WORK

As software maintenance takes up a major part of software projects, we are interested in investigating which documentation is more helpful in supporting maintainers when they have to maintain source code. This paper specifically presents a family of experiments that were carried out to investigate whether using either FD or RE UML diagrams (class and sequence diagrams) improve the maintainer’s performance when modifying source code. We therefore wished to ascertain whether it is beneficial to build UML diagrams in the initial phases of development and kept them up-to-date (which consumes more effort) or whether on the contrary it is better to rely on UML diagrams obtained by reverse engineering the source code.

The original experiment (E-US1) was carried out by 40 second-year students on a Master’s Degree in Computer Science. The first replication (R-US2) was conducted by 51 students enrolled on the same degree and at the same university as the experiment. The second replication (R-UB) was performed with 78 second-year Laurea Degree students in Computer Science at the University of Bari.

The statistical results, and specifically the statistical test and the descriptive results, show a tendency towards obtaining better results when using UML diagrams (concretely class diagrams), that were hand-made during the design phase. Based on the qualitative results of the post-experiment survey, it is also important to note that the subjects preferred FD diagrams when understanding and maintaining a system. This is true even though their performance is not much better with FD diagrams in some cases, in comparison to their performance with RE diagrams. Because software maintenance is a human-based process, this highlighting of maintainers’ perceptions in favor of using FD diagrams is very important.

In addition, we found that subjects who received RE diagrams experienced more difficulties when reading the diagrams, especially the sequence diagrams. Although the subjects who received FD diagrams felt that sequence diagrams were highly useful, as they expressed in the post-experiment survey, only a small number of subjects actually used the diagrams for the tasks in the experiment. In the case of the RE diagram group, the subjects did not use RE sequence diagrams. These subjects point out that these RE sequence diagrams are not very useful due to their low level of readability/high level of details. Even though the results of the family of experiments are not homogenous in all the experiments, the evidence shown in the MEffic test in R-US2, the results of the meta-analysis and the subjects’ opinion extracted in the post-questionnaire survey point in the direction that software maintainers prefer to use FD UML diagrams over Reverse Engineered UML diagrams.

These results give us grounds to encourage software developers, albeit with caution, to follow a model-centric approach in projects with novel maintainers and
small-sized systems related to well-known domains. Nonetheless, other contexts should be explored in order to reaffirm the results in an industrial context by carrying out further replications with professionals, considering more complex systems related to non-well known domains.

The findings obtained imply beginning the development of a software system by creating UML diagrams, and in addition to keep them up-to-date, thereby making it easier to perform maintenance tasks. Class diagrams are important artifacts which are widely used and highly appreciated by maintainers. However, there is a doubt as to whether the documentation, and UML as part of it, will be maintained as the system evolves when a model-centric approach is used (Petre, 2013). This goes against an effective use of the diagrams, a fact that suggests that we should recommend companies to keep their diagrams up-to-date and thus help their maintainers to perform the required tasks efficiently. The results obtained are therefore useful for all those companies that exploit this notation as a support for software maintainers when performing maintenance tasks.

The recommending on the use of Forward Designed UML diagrams (at least the class diagrams) assumes that UML diagrams were created during software development, and kept up-to-date during software maintenance. This requires an additional effort in the software lifecycle in comparison with the automated generation of UML diagrams from source code through reverse engineering techniques. Currently it seems that reverse engineering techniques are not able to abstract away less important information from the source code. Hence diagrams obtained via RE also require manual effort for simplifying them. The empirical investigation of whether this extra effort is worthwhile is still pending, but would provide us with knowledge regarding the return on investment of UML modeling in software maintenance. This topic has not been explored in detail, and this is necessary if we are to discover whether UML will have a widely acceptance in industry and how UML can gain a widely adoption in industry.

**Acknowledgements.** This research has been funded by the GEODAS-BC project (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01). The authors would like to thank the students who have cooperated in the experiments.
CHAPTER 5. WHAT IS THE PERCEPTION OF INDUSTRY PROFESSIONALS AS REGARDS THE VALUE OF USING UML IN SOFTWARE MAINTENANCE?

Chapter 5: What is the Perception of Industry Professionals as Regards the Value of Using UML in Software Maintenance?

ABSTRACT

This paper presents the findings of a survey on the use of UML in software maintenance, carried out with 178 professionals working on software maintenance projects in 12 different countries. As part of long-term research we are carrying out to investigate the benefits of using UML in software maintenance, the main objectives of this survey are: 1) to explore whether UML diagrams are being used in software industry maintenance projects; 2) to see what UML diagrams are the most effective for software maintenance; 3) to find out what the perceived benefits of using UML diagrams are; and 4) to contextualize the kind of companies that use UML documentation in software maintenance. Some complementary results based on the way the documentation is used (whether it is UML-based or not) during software maintenance are also presented.

Keywords: UML, Software Maintenance, Survey.

5.1. INTRODUCTION

UML (OMG, 2010) has become the de facto standard modeling notation used as a graphical notation to complement software documentation. It would therefore be useful for the software industry to study whether or not the use of UML benefits software maintenance, particularly because any type of investment must be justified from an economic point of view; i.e., there should be a payback at a later phase. In the context of software projects, investment in modelling should thus be justified by benefits (such as improved productivity and better product quality) that can be gained later, during software development or maintenance.

To the best of our knowledge, there are only three empirical studies on the impact of UML documentation on software maintenance in industry. Dzidek et al. (Dzidek et al., 2008) presented an experiment using 20 professional developers as subjects. Scanniello et al. (Scanniello et al., 2010) showed the results of an exploratory survey to investigate the state of the practice regarding the use of UML in software development and maintenance, with 22 employees at Italian companies. Fernández-Sáez et al. (Fernández-Sáez et al., 2013c) presented an industrial case study performed in a large ICT (Information and communication technologies) department, in which 20 ICT professionals were interviewed. Their aim study was to investigate the use of UML diagrams during software maintenance. Summing up the results of these 3 studies, it would appear that this notation is frequently used, at least in Italy. This might be because using UML during maintenance seems to be beneficial in terms of software quality, even though no time is saved.

Although there is some encouraging evidence concerning the benefits of using UML during software maintenance in industry, it is scarce and based on a small population. It would therefore be useful to go one step further and collect a larger population of evidence from industry. Case studies in industrial contexts typically take a long time; it is difficult to obtain a large population of projects in industry that provide appropriate data. We have therefore attempted to bridge this gap by carrying
out a survey of 178 ICT professionals from 12 countries (across the globe); that survey will be presented in this work as a first approach to the status of the current industrial environments. We decided to use this method because surveys are well-established social science techniques that can be used to gather information and opinions from a large population known to be representative of a target population (Pinsonneault and Kraemer, 1993). Online-surveys may have some limitations, such as sampling bias and difficulties in designing clear, unbiased and unambiguous questionnaire items. But we have attempted to mitigate these as far as possible, by involving external researchers and some employees in the review and improvement of the experimental material, and by taking into account similar studies in the software engineering community.

The main goal of this survey is to address the following research questions (RQs):

- RQ1: Is UML documentation used to support software maintenance in industry?
- RQ2: What are the perceived benefits of using UML during software maintenance?
- RQ3: What kinds of companies use UML documentation during software maintenance?
- RQ4: What subset of UML diagram types has been demonstrated to be most effective during software maintenance?

The remainder of the paper is organized as follows: Section 5.2 describes the main steps of the survey design, while the results of the survey are presented in Section 5.3. The threats to validity are set out in Section 5.4. Finally, our conclusions and future work are presented in Section 5.5.

5.2. SURVEY DESCRIPTION

The survey was designed and reported by following the recommendations provided in (Kitchenham and Pfleeger, 2002).

5.2.1. Goal and research questions

The main goal of this survey is to address the RQs presented in Section 5.1.

5.2.2. Target population, sample identification and recruitment strategy

Our target population consisted of practitioners who have worked on maintenance projects, whether or not they have used UML. We considered ICT companies that develop, maintain or sell software as a principal part of their business, or companies focused on other business but with a large ICT department. The selection of the companies (sampling) was conducted by using the network contacts of the research groups of the authors of this paper who conducted the survey. Each author defined his or her own list of contacts, which included:
employees at companies involved in research projects with the authors’ universities, or that host students from the authors’ universities for internships or thesis projects;
- the authors’ former students, now employed at software companies;
- researchers from other universities with whom the authors have collaborated;
- people from professional networks or companies included in public-private research of which the authors’ universities are members.

Upon receiving a filled-in survey, we asked the respondent to provide us with more contacts. A single list of 585 contacts was eventually obtained and used to distribute the survey. We also advertised the survey in software maintenance communities on the Internet, on sites such as the International Software Engineering Research Network, (which is for people who follow the International Conference on Software Maintenance), or the site of Software Maintenance and Reengineering.

5.2.3. Survey structure

The survey was structured in blocks which grouped the questions into four topics:

1. **Demographic information**: this refers to information about the person replying such as: gender, educational qualifications, country in which they work, role in the company, experience in ICT and experience in software maintenance. This block of 7 questions helped us to contextualize the responses obtained.

2. **Organizational information**: the objective was to characterize the respondent’s company. In particular, we collected information concerning: the size of the ICT department, stability of the maintenance team, or whether the company is geographically distributed or co-located. This block contained 6 questions designed to answer part of RQ3.

3. **Project information**: this refers to information regarding the most typical projects carried out in the company. The block included questions related to items such as: size of systems maintained, size of maintenance teams or type of maintenance carried out, and contained 4 questions related to RQ3. It is important to define the types of maintenance mentioned in this paper. They were divided into the following categories (Swanson, 1976): 1) **Corrective maintenance tasks**, i.e., those related to fixing a bug, 2) **Adaptive maintenance tasks**, i.e., those related to the changes made to the hardware/software platform, interface or requirement in order to improve performance or conform better to the law, or changes in the operative context; and 3) **Evolutive maintenance tasks**, i.e., those related to the development of new functionalities or functional/technical requirements requested by a customer.

4. **Process information**: this consisted of questions in which we asked whether the respondents create UML during software development, and if so, of what type; we also asked about their use during software maintenance. This block contained 11 questions related to RQ1, RQ2 and RQ4.
5.2.4. Survey design

To address the research questions formulated, we drew up a survey consisting of 4 blocks of questions, with 28 questions in all. Some questions were not presented to all individuals, as they were determined by the responses provided to other questions (i.e., conditional ones). Each person therefore answered a maximum of 22 questions. The electronic copy of the survey and the questionnaire flow is available online at:

http://alarcos.esi.uclm.es/ShortSurvey-UML-Maintenance/

Most of the questions were measured using Likert scales, and a few others were measured with nominal scales, but they were all closed questions. Some of them also included a space for extra information, however. To avoid bias, the questions were ordered in such a way that the answer to one question would not affect the answers to the following ones. Though originally designed in English, Spanish and Italian versions were also used.

5.2.5. Survey construction and execution

The procedure followed consisted of the following steps:

1. An initial set of questions was selected by using similar surveys (such as those in (Agner et al., 2013; Tomassetti et al., 2012)) as a basis and tailoring them to our goals. A list of possible contacts was created by following the recruitment strategy explained above.

2. A pilot study with five industrial ICT professionals from an Italian company was performed before the survey was made available online. This was to refine it and to reduce any ambiguities, and minor changes were then made to the survey.

3. The survey was online from February to April of 2013, using Survey Monkey (SurveyMonkey, 1999).

4. Contacts were invited (via email or phone) to participate the study. A reminder was sent to those who had committed to completing the survey, but who had not returned it by the end of March.

5. After the surveys had been collected, analyses were performed, aiming to answer the research questions. Data analysis was based on a quantitative analysis focusing mainly on descriptive statistics and percentages of the information collected.

5.3. RESULTS

A total of 268 ICT practitioners of the 585 directly contacted showed interest in responding to the survey. We filtered some of these, because, although they were interested in collaborating, they did not have the profile intended for this survey; i.e., they did not work in software maintenance. In the end, 178 responded to the survey during the two months it was online. This result is significant because of the difficulty normally involved in obtaining such a large quantity of individuals suitable for making up a target population. Importantly, no money or other incentives were given to the respondents, making the very high number of responses for a study of these
characteristics in ICT environments even more surprising. We cannot state how many different companies these 178 people represent, as for reasons of privacy we did not ask the respondents to indicate their company. Please bear in mind that the responses to all the questions, summarized in the following subsections, do not add up to 178 in all cases, since some people did not answer all the questions (because of some conditional questions).

5.3.1. Overview and descriptive statistics

When analyzing the demographic information from the survey, we attempted to describe the respondents’ profiles. 78% of them (139) were male and 22% were female (39), which was the proportion expected based on our personal perception of typical proportions for ICT. The countries in which the participants work are very varied: Afghanistan (1), Algeria (1), Austria (2), Canada (1), China (4), Finland (1), India (2), Italy (110), Mexico (1), Netherlands (18), Spain (23) and Uruguay (14).

The majority of those taking part have a high educational level with a Master’s Degree (43%), or a medium high level with a Bachelor’s Degree (33%); 4% have a researcher profile (they have PhD studies) and 17% have high school studies only. These percentages allow us to state that the sector is mature in terms of skilled professionals.

In terms of experience, the majority of the respondents (71%) are experienced professionals with more than 5 years in the area of ICT, and only 2% have less than 1 year of experience in this field. The other 26% of the survey participants possess between 1 and 5 years of experience. If we focus on experience in the field of software maintenance, then the percentages change: over half the respondents have more than 5 years of experience; only 6% have less than 1 year. The results lead us to assume that some experience is needed in the ICT field in general before working in the software maintenance field. This may be due to the need to have sufficient experience in understanding how systems are built before being able to modify them. We could have excluded those respondents who did not have very much experience in software maintenance; we decided against this, in order to obtain a real representation of industrial workers. The role most frequently played by the participants is that of programmer/coder (34%), followed by software analyst (19%), and project manager (15%). The remaining roles (business analyst, designer, software architects, software testers, etc.) are performed by less than 8% each. 66% of the respondents are currently working on software maintenance, while the others (34%) have worked on it in the past. In most cases (61%), those who replied perform maintenance on software which was developed by the same company, as against the 37% who maintain software developed by third parties.

The demographic results are consistent with those found in (Grossman et al., 2005), i.e., most UML users are highly-educated and experienced. They also play a variety of roles, but most of them are software developers.
If we filter out those who do or do not use UML diagrams during software maintenance by role (Figure 5.1), we can state that the roles that use UML diagrams most frequently are software architects (as expected, because they create them), followed by software analysts and project managers. Project managers may be UML consumers (Petre, 2013). Those who use UML diagrams least are business analysts, software testers and maintenance engineers. It is sometimes difficult to know whether programmers use the UML diagrams provided by the architects (to “measure” whether the investment of creating the UML diagrams provides any kind of payback). We would therefore like to stress that almost one third (27%) of the programmers use UML diagrams for software maintenance tasks.

![Figure 5.1. Use of UML diagrams during software maintenance per role.](image1)

We classified the people who do or do not use UML diagrams during software maintenance by educational level (Figure 5.2). It “seems” that a higher educational level leads to a greater use of UML diagrams (except in the case of PhD students, who may have specialized in topics that are very unlike software modeling). This

![Figure 5.2. Use of UML diagrams during software maintenance by educational level.](image2)
assumption was made after discarding the results of those groups with low representativeness, due to their low generalizability.

5.3.2. RQ1: Is UML documentation used to support software maintenance in industry?

We asked the respondents about what type of documentation they use during software maintenance. The majority use artifacts that are based on textual documentation (66%) and on the source code by itself (70%), but 40% of those surveyed use a graphical notation (26% use only UML, 9% another notation), to support the design of the changes related to software maintenance tasks. This means that, when a graphical notation is used, UML is used in 72% of the cases in comparison to other notations. This is more or less the same proportion as in the results obtained by Hutchinson et al. (Hutchinson et al., 2014) in their survey regarding MDD (Model Driven Development) and the different notations used for it. The percentage of UML use obtained is also similar to that in the results obtained by M. Petre (Petre, 2013) in her study on the use of UML (in general, not only focusing on maintenance). But these two results are contrary to the results obtained by Scanniello et al. (Scanniello et al., 2010), which reveal that 75% of the respondents (all from Italian companies) use UML. As stated previously, 20 respondents (9%) use a different graphical notation (like those in the results obtained in (Agner et al., 2013)). There are also 8 respondents (5%) who use a combination of UML and other graphical notations. This reinforces the results obtained by Hutchinson et al. (Hutchinson et al., 2014), i.e., the use of different notations is not an unusual practice, because most notations are not selective. The other graphical notations used are the following: BPMN, E/R, SysML, FSP-SPEM, database designs, Archimate, screenshots or domain specific languages, but the majority does not use formalized notations ("boxes and arrows"). Most of them coincide with the languages mentioned in the survey of Hutchinson et al. (Hutchinson et al., 2014).

Those who do not use any graphical notation but who do employ textual documentation represent 33% of the respondents. 14% do not use any complementary documentation (graphical or textual) to maintain source code, i.e., they only employ the source code as documentation. This is surprising, since it is additional information to the source code and requires an extra investment for its creation. But source code and its comments are the most important artifacts for understanding a system that is to be maintained (de Souza et al., 2005).

From here on we shall use the terms “UML group” for those who stated they have UML diagrams available as part of their maintenance documentation, and “non-UML group” for those who do not have UML diagrams in their documentation.

We asked the UML group how often they do not consult software documentation and work directly with source code (Table 5.1).

Those who indicated they never discard the UML diagrams to work directly with source code form 8% of the UML group. It may be that they do not use source code
because of their roles: project managers, software analysts, designers and software architects.

### Table 5.1. Frequency of use of source code only.

<table>
<thead>
<tr>
<th>Frequency of Use</th>
<th>UML Group</th>
<th>Non-UML Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Never</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Some-times</td>
<td>22</td>
<td>37</td>
</tr>
<tr>
<td>Often</td>
<td>20</td>
<td>43</td>
</tr>
<tr>
<td>Very often</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>Always</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

Almost half of the UML group (45%) do not always consult the documentation, and work directly with source code. They justified this way of working ideas as follows:

- There is no documentation in the project.
- Documentation does not always describe the observed behavior of the application.
- It is a useful way to handle defects.
- Source code is always the most reliable documentation.
- When the UML diagrams are created a posteriori (legacy systems) they do not have enough detail.
- UML is not useful when the maintenance task needed is an evolutive task of the system with low impact on the rest of it.
- Those who often discard the documentation and work directly with source code (41%) do so when:
  - The software maintenance task is small, and it is faster to do it directly in the source (doc. is not needed).
  - The system under maintenance is well-known by maintainers. It is recognized that the documentation is not aligned with the source code.
  - There is a lack of time.

In the case of the non-UML group (Table 5.1), 7% indicated that they never discard the documentation to work directly with source code.

33% of the non-UML group do not always consult the documentation and work directly with source code. They discard documentation, for the following reasons:

- Sometimes source code is self-explanatory. When the system concerned is very old, the documentation may very well no longer describe the current situation.
- Code sometimes contains more details than documentation.
- Documentation is not necessary for testing.
- There is a lack of tooling for the synchronized updating of source code and documentation.
- Those who discard the documentation and directly use source code more often (53%) do so because:
  - The documentation documents are very long. The documentation is not properly structured, or not very accessible (parts in emails, for example).
- There is a good deal of knowledge embedded in source code which is not present in the documentation.
- Systems are well-known.
- Documentation might be not updated (only when the customer requires it).
- Documentation does not exist in some cases (especially in legacy systems).
- The maintenance tasks are small or corrective.
- Time pressure exists.

5.3.3. RQ2: What are the perceived benefits of using UML during software maintenance?

The UML group was asked why they use UML diagrams. Some possible reasons were presented to them.

The majority of the UML group (55%) believes that the UML based diagram documentation provides added value for software comprehension and defect detection. One recurrent argument (41%) for using UML is the idea that UML based diagram documentation reduces the time needed for software comprehension and defect detection.

Some of them (31%) also think that UML based diagram documentation outperforms the other available standards, diagrams and models. There are also a few (29%) that use it because it has been adopted by their companies, i.e., they are “forced” to do so. A subset of survey respondents (27%) thinks that the UML based diagram documentation reduces maintenance costs.

A minority (8%) use UML because they do not know of any other alternatives. Some of them (8%) also justified the use of UML as follows:
- It is a standard, i.e., it is not ambiguous.
- It is familiar for most developers.
- It is an easy communication model that makes it easier to review the development activity.
- It is easy to understand for technical and non-technical people, because it has different views.

The non-UML group was similarly asked why they do not use UML:

The option chosen most frequently was “I have to use the standards, diagrams and models adopted by my company” (33%). The next reason for not using UML is because the non-UML group prefers working directly with source code (23%). This percentage is double that obtained in the survey of Scanniello et al. (Scanniello et al., 2010). On the other hand, 17% of the non-UML group believes that time spent on UML diagram comprehension is not compensated by the benefits of using UML. This percentage is much lower than the 50% of people surveyed in (Torchiano et al., 2011) who excused their non-use of modeling by saying that models require too much effort.

Finally, it is strange that some of those responding do not use UML simply because they are not familiar with it (14%). These data coincide with the results of (Agner et
al., 2013) and (Nugroho and Chaudron, 2008). Similarly, 10% of non-UML group believes that the UML based diagram documentation does not add enough value to software comprehension.

A minority (2%) thinks that the standards, diagrams and models used in their companies are better than UML based diagram documentation. Some of them (18%) also argued for the non-use of UML, giving the following reasons:

- It is difficult to manage versions of diagrams. This contradicts the results obtained in (Hutchinson et al., 2014), in which it was claimed that 50% of companies used versioning tools for modeling (in the context of MDD).
- Legacy systems do not usually have UML diagrams in their documentation.
- UML diagrams are not usually maintained.
- Final customers do not like UML diagrams.
- There is minimal use of documentation, in general.
- When the development starts, the requirements are unclear.
- Diagrams are used for personal purposes but not stored as documentation.

When the questionnaire was created, we took into account that one of the responses to the previous question might be that maintainers use software documentation (containing UML diagrams, or text or other graphical notations) only infrequently. We thought that the effort of consulting the UML/documentation might be great, thanks to that low rate of use. A question about this was therefore added to the questionnaire (Table 5.2).

For the UML group, the effort of consulting the UML diagrams is almost always less than 20% of the total effort made to maintain the system (only 16% of the UML group disagreed with this statement). For the non-UML group, the proportion of those who spent more than 20% of the effort in consulting the documentation is slightly lower. This is because over 1/4 of them use more effort in consulting the documentation, and it may explain why more maintainers in the non-UML group than in the UML group do not use the documentation.

<table>
<thead>
<tr>
<th></th>
<th>&lt;10%</th>
<th>11%-20%</th>
<th>21%-30%</th>
<th>31%-40%</th>
<th>&gt;40%</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML group</td>
<td>22</td>
<td>19</td>
<td>7</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Non-UML group</td>
<td>46</td>
<td>38</td>
<td>20</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

5.3.4. RQ3: What kinds of companies use UML documentation during software maintenance?

A “Company” is the final result of several factors, such as organization type, dimension, business domain, type of projects carried out and processes in use. In the survey we attempted to investigate each of these aspects with a set of focused questions.
We asked those surveyed whether UML use is closely tied to the software development methodology used for software maintenance, to discover if this is indeed the case (Figure 5.1).

Only when the development methodology is RUP–like are there more respondents from the UML group than the non-UML group (rate of 2). This proportion was expected, given the particularities of this development process. On the other hand, when following waterfall methodologies or Product Line developments, the proportion of UML users falls drastically (rate of 0.57 and 0.25, respectively).

In the case of other methodologies, namely agile approaches, Prince2, ISO/IEC 29110, MoProSoft, ITIL V3, Polarion, ASAP (SAP), Spiral, there is one UML user per each 3 maintainers.

Some of those taking part also mentioned that they use one or more of their own methodologies, depending on the project.

The modeling tool used to maintain/modify the UML diagrams is an important factor when deciding whether to use an UML based software development process. There are different types of tools with different benefits: licensed tools (which implies an investment but also payback with possible training, customizations, etc.) vs. open tools, or specific tools for modeling in UML (which check the correctness of syntax) or general modeling tools (these are more “accessible”).

24% of the UML group does not use a modeling tool: 2% because they manage diagrams on physical paper or blackboards and the diagrams are not digitalized, while the rest (22%) do not manage diagrams because they are not modifiable images (.jpg,
bmp, .pdf, etc.). This is a higher percentage than that obtained in (Agner et al., 2013), whose results revealed that only 6.4% used modeling tools.

The 73% UML group, which uses tools to modify the UML diagrams, might do so using a single tool, or have more than one available for the same purpose.

It is quite surprising that one of the most frequently-used tools is Microsoft Office Visio (29%), which is non-specific to UML design. It is true that it contains a toolbar for UML design, but it does not check the basic correctness of the UML diagram, i.e., the syntax is not checked. The tool thus allows any element to be connected with another one, or even elements from one diagram to be introduced in another (for example, actors in a class diagram). One reason for the frequent use of this tool might be that a lot of companies work with it for other purposes; employees already have the tool installed, making it easier to use it and to share diagrams.

Enterprise Architect, which is a very comprehensive licensed tool, has the same percentage of use (29%). It contains the option of producing Reverse Engineering (RE) UML diagrams, compared to the next most widely-used tool, which is the open tool StarUML (14%). Others mentioned are (with 2% to 7% each): Rational Rose, ArgoUML, IBM Software Architect, DIA, Visual Studio, Gliffy and UML designer (a plugin for Eclipse).

It is also worth noting that 83% of the respondents belong to geo-distributed companies. But how often is the maintenance team geo-distributed? Half of the participants work in maintenance teams that are geo-distributed, while the other half are in co-located teams (Table 5.3). We see, however, that the proportion of UML use is slightly lower (9% less) in the case of co-located maintenance teams.

<table>
<thead>
<tr>
<th>Geo-distributed maintenance team</th>
<th>Single-site Maintenance team</th>
</tr>
</thead>
<tbody>
<tr>
<td># respondents (in total)</td>
<td>%</td>
</tr>
<tr>
<td>Use UML in maintenance</td>
<td>30</td>
</tr>
<tr>
<td>Do not use UML in maintenance</td>
<td>56</td>
</tr>
</tbody>
</table>

Table 5.3. Relationship between geo-distribution of the maintenance teams and use of UML in software maintenance.

One of the purposes of using UML diagrams is to improve communication between stakeholders (Fernández-Sáez et al., 2013c; Tryggeseth, 1997). When a company is geo-distributed, this factor becomes critical. In both cases, fewer companies use UML during maintenance, regardless of their locations. The proportion of companies that use UML diagrams is, surprisingly, a little higher (9% extra) for those companies that are geo-distributed in comparison to those that are co-located (
Table 5.4). This could be because co-located teams are more standardized as regards development methods and tools.

Table 5.4. Relationship between geo-distribution of companies and use of UML in software maintenance.

<table>
<thead>
<tr>
<th></th>
<th>Geo-distributed company</th>
<th>Co-located company</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># respondents (in total)</td>
<td>%</td>
</tr>
<tr>
<td>Use UML in maintenance</td>
<td>40</td>
<td>28%</td>
</tr>
<tr>
<td>Do not use UML in maintenance</td>
<td>105</td>
<td>72%</td>
</tr>
</tbody>
</table>

We also studied the influence of company size on the use of UML diagrams. We measured the size of companies, using their number of employees. The majority of those responding (66%) belong to very large companies, with more than 250 employees. Large companies (with 50 to 250 employees) use a higher proportion (40%) of UML diagrams than those of other sizes (Figure 5.4). In the rest of the categories, the use is less than 30%.

Figure 5.4. Relationship between size of company and use of UML in software maintenance.

We then looked into the influence of the size of the ICT departments on the use of UML (see Table 5.5). Companies with small ICT departments, i.e., with fewer than 10 employees, use fewer UML diagrams (around 16%) than bigger ICT departments (from 26% to 39%). This was as expected, since one of the main reasons for using UML diagrams is communication between team members; there is expected to be less need for codified design knowledge in small ICT departments.
Chapter 5: What is the Perception of Industry Professionals as Regards the Value of Using UML in Software Maintenance?

Focusing on team size (Table 5.6), we considered small teams to be those with fewer than 3 people, medium to be those with between 5 and 9 people, large to be those with 10 to 49 people, and very large to be those with 50 or more people. The majority of the respondents work in small (44%) or medium (35%) sized teams. Team size does not appear to be an influential factor in the use of UML during software maintenance. It is also important to note that, although the ICT departments are very large, team size does not tend to be correspondingly large. This is why there are more respondents from small teams than from large ones. What is more, small teams use fewer UML diagrams than large ones, because they have facilities for face-to-face meetings, and so need less supporting documentation.

Table 5.5. Relationship between size of ICT department and use of UML in software maintenance.

<table>
<thead>
<tr>
<th>Size of ICT Department</th>
<th>Use UML in maintenance</th>
<th>Do not use UML in maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;10</td>
<td>3 (16%)</td>
<td>16 (84%)</td>
</tr>
<tr>
<td>[10-50]</td>
<td>10 (38%)</td>
<td>16 (62%)</td>
</tr>
<tr>
<td>[50-250]</td>
<td>12 (39%)</td>
<td>19 (61%)</td>
</tr>
<tr>
<td>&gt; 250</td>
<td>26 (26%)</td>
<td>73 (74%)</td>
</tr>
</tbody>
</table>

The type of maintenance team was also studied. In most cases (80%) the respondents belong to stable maintenance teams whose objective is to directly develop or maintain software. In the remaining cases (20%), the team was created when needed. Team stability does not seem to be an influential factor in the use of UML; the proportion is the same for both kinds of teams (29%).

Table 5.6. Relationship between team size and use of UML in software maintenance.

<table>
<thead>
<tr>
<th>Team Size</th>
<th>Use UML in maintenance</th>
<th>Do not use UML in maintenance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>25 (33%)</td>
<td>51 (67%)</td>
</tr>
<tr>
<td>Medium</td>
<td>14 (24%)</td>
<td>45 (76%)</td>
</tr>
<tr>
<td>Big</td>
<td>9 (35%)</td>
<td>17 (65%)</td>
</tr>
<tr>
<td>Very Big</td>
<td>3 (50%)</td>
<td>3 (50%)</td>
</tr>
</tbody>
</table>

With regard to the size of the systems maintained, these are classified depending on their number of Lines of Code (LoC). A small system is one with fewer than 10,000 LoC (10% of respondents); a medium system has between 10,000 and 100,000 LoC (38%); a large system might have between 100,000 and 500,000 LoC (33%), while a very large system would have more than 500,000 LoC (19%) (Scanniello et al., 2010). A higher use of UML diagrams was expected in projects that maintain larger systems. One of the reasons put forward for using UML (or models) is that it helps manage large and/or complicated systems. The results obtained show that UML diagrams gain popularity when the team has to maintain a very large system (Table 5.7), but differences are not too great (from 20% to 39%).

The type of maintenance most often performed is evolutive, since 82% of the respondents do it frequently (i.e., 60 often, 59 very often or 27 always), followed by corrective tasks. The adaptive maintenance tasks are done less often (33% of the respondents never do so, or do so rarely). These results are similar to those obtained by Souza et al. (de Souza et al., 2005), who state that the most frequently-used maintenance is evolutive.
Chapter 5: What is the Perception of Industry Professionals as Regards the Value of Using UML in Software Maintenance?

Table 5.7. Relationship between size of system maintained and use of UML in software maintenance.

<table>
<thead>
<tr>
<th>Use UML in maintenance</th>
<th>Small</th>
<th>Medium</th>
<th>Big</th>
<th>Very Big</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use UML in maintenance</td>
<td>4 (29%)</td>
<td>19 (31%)</td>
<td>18 (33%)</td>
<td>10 (39%)</td>
</tr>
<tr>
<td>Do not use UML in maintenance</td>
<td>10 (71%)</td>
<td>43 (69%)</td>
<td>37 (67%)</td>
<td>26 (61%)</td>
</tr>
</tbody>
</table>

We studied whether the business sector type of the company influences the use of UML during maintenance. The categories used to classify companies by business sector were the following: Finance (0% of respondents); Telecommunications (2%); Manufacturing (3%); Service Provider (22%); SW Development, Maintenance and Service (69%); or other sectors (4%). Other sectors included education, medicine, or humanitarian companies. If we focus on those respondents who belong to “Software companies”, 35% use UML. As expected, there are other sectors not directly dedicated to software development (they create software as a resource for their companies, but their main business focus is on other items) that do not seem to use UML diagrams. In contrast, a low proportion (13%) of those working in the “Service Provider” sector uses UML diagrams in maintenance, while 40% of those employed in “Manufacturing” companies use the UML diagrams. It is surprising that companies not directly dedicated to software development use UML diagrams (13% of Service Providers, and 40% of Manufacturing Companies surveyed, although these percentages are calculated based on low representativeness). This implies that UML is being used not only in the context of software factories.

Figure 5.5. Relationship between role of software in the company and use of UML in software maintenance.

More than half the respondents (66%) are employees of companies in which the company’s business is the software itself; i.e., they work in ICT companies (Figure
5.5. Another large percentage of the participants (36%) work for companies in which the software is a strategic resource to support their business, but they focus on producing other kinds of products, or on providing other kinds of services. Only 5 respondents (3%) worked for companies in which software is a marginal element. The percentage of UML usage is higher in the case of those companies in which the software is the main business element (33%), compared to those in which the software is a supporting element for the company’s business (24%).

5.3.5. RQ4: Which subset of UML diagram types has been demonstrated to be most widely-used and effective during software maintenance?

The UML diagrams that are most frequently used during software maintenance (Figure 5.6) are class diagrams (61% of UML group), use case diagrams (45%), sequence diagrams (41%) and activity diagrams (33%). These results are similar to those of other surveys (Agner et al., 2013; Dobing and Parsons, 2006; Grossman et al., 2005; Hutchinson et al., 2014) in which class, activity, use case and sequence diagrams are part of the top 4 UML diagrams used. The diagrams used least are the collaboration diagrams (perhaps because they are equivalent to sequence diagrams), composite structure diagrams, interaction view diagrams and timing diagrams (all of which are UML 2.0 diagrams and less well known).

![Figure 5.6 UML diagrams used during software maintenance.](image)

The origin of the UML diagrams is usually the development phase (94%); i.e., they are not created especially during the maintenance, except where UML diagrams are not available from the software development; then they are created specifically during maintenance (3 respondents; 6%). It is also quite surprising that 34 of those replying (44%) said that UML diagrams are available from the development phase, but that they do not use them during software maintenance at all. This might be the result of a divergence between the diagrams and the source code.
The UML diagrams produced during SW development are summarized in Figure 5.7. Curiously, some survey respondents (7) considered UML diagrams to be other types of diagrams, such as: Business Process Models, Data models, DDL, Test case, Navigation diagrams, and user interface prototypes. It does not therefore appear to be clear which are UML diagrams and which are not.

The UML diagrams created during software development.

When the UML diagrams are created expressly for the maintenance tasks, two approaches might be followed: 1) the diagrams could be human-based diagrams, i.e., they are created manually using a forward design approach, and 2) they could also be machine-based diagrams, i.e., the diagrams are created automatically by software using a Reverse Engineering (RE) technique.

If the responses from those who do not know the origin of the UML diagrams (3%) are discarded, the majority of the UML groups use human-based diagrams (81%). Of the rest, those who use UML diagrams generated using any tool with a RE technique, 3% use pure RE diagrams (completely automatic UML diagrams). 19% employ RE UML diagrams reviewed by humans (which could be considered as semiautomatic diagrams). In relation to this, the results of [18] show a tendency toward better results being obtained when using UML diagrams (class diagrams, specifically) that were hand-made during the design phase. The results from (Fernández-Sáez et al., 2015a) also revealed that maintainers using RE diagrams experienced more difficulties when reading the diagrams. Most companies surveyed therefore use the “more understandable” UML diagrams. Maintainers do not always employ the available
documentation and work directly with the source code; even if the documentation is available, it is not used.

5.4. THRUGHTS TO VALIDITY

We shall now analyze the main potential threats to the validity of the survey presented here:

- **Internal validity**: the main issues affecting the internal validity of our study concern the framing and sampling of the participants. Our recruitment strategy could have incurred a possible selection bias (for example, a high probability of profile similarity among the respondents). However, we believe that the issues analyzed are not affected by this threat; the sample size is large and there is variety in the roles and nationalities of respondents. Another threat derives from the channel used to survey maintainers; the questions may have been answered by respondents who did not have the knowledge required to do so. We attempted to address this issue when we defined the protocol of the survey: we explicitly required the survey to be filled in by ICT professionals (the target roles were specified) involved in the maintenance of software systems (the definition of maintenance was provided). Another negative factor could have been the difficulty involved in understanding the questions (e.g., ambiguous, unclear, not well-formulated), and the respondents’ motivations might also have affected the answers and thus the survey results. In web-based surveys the sampling procedure makes it possible to select duplicate units; one person might answer the survey more than once. We addressed this threat by using a system consisting of a single link per person. The reader may also object that the companies within our industrial network might also have influenced the internal validity, and that several people from the same company may have answered the survey, thus biasing the results.

- **External validity**: To interpret the results we obtained correctly, it should be borne in mind that, although the demographics of our sample are fairly diverse, generalizing our results to the entire population may not be appropriate. In our survey, the companies belonged to a variety of domains and covered different company sizes in various countries throughout the world; we cannot be certain that our sample is representative of the ICT industry in general, however. These threats are always present in industrial surveys.

5.5. CONCLUSIONS AND FUTURE WORK

This paper reports the findings of a survey on the use of UML in software maintenance to which 178 ICT professionals responded. The main findings, grouped by RQs, are:

- **RQ1: Is UML documentation used to support software maintenance in industry?**
  
  59% of those surveyed use a graphical notation (43% UML; 16% another notation) as a complement in trying to understand the system that will be maintained. In
contrast, 28% of the respondents use only source code; they consider that source code and its comments are the most important artifacts in understanding the system to be maintained. It is quite surprising that maintainers do not always use the UML diagrams that are available from the development phase. This might be due to problems of a lack of synchronization caused by non-updated diagrams.

• **RQ2: What are the perceived benefits of using UML during software maintenance?**

The main reasons for using UML are that less time is needed for a better understanding of the system under maintenance; this improves defect detection. Reasons given for not using UML are that maintainers follow other standards provided by their companies or that they prefer to work directly with source code. Moreover, when UML diagrams are available as part of the documentation, it takes less effort to consult them. That might explain why more maintainers do not use the documentation in the group whose documentation contains UML than in the group whose documentation does not.

• **RQ3: What kinds of companies use UML documentation in software maintenance?**

The size of the maintenance team appears to influence the use of UML. Larger teams use UML diagrams more frequently, proportionately, perhaps because of the improvement to the understanding of the system provided by UML diagrams and due to the need to share/communicate knowledge in this kind of teams. The size of the system being maintained also seems to be an influential factor. The results obtained show that UML diagrams are extremely popular when the team has to maintain a very big system. This seems to be logical, since one of the reasons put forward for using UML (or models) is that it helps manage large and/or complicated systems.

Regarding additional results for characteristics of companies, we should note that geo-distributed maintenance teams are common (50% of the cases), although this is not influential in UML use. Moreover, the most common type of maintenance is evolutive. It is also noteworthy that maintenance tasks seem to be carried out by those who already have experience in ICT, i.e., they have worked in development before maintenance (first they learn to create it, and then they learn to change it). Moreover, and surprisingly, Visio is the most commonly-used tool for UML modeling, although it is not specific to UML. It is followed by Enterprise Architect (licensed tool) and StarUML (open source tool).

• **RQ4: Which subset of UML diagram types has been demonstrated to be most widely-used and effective during software maintenance?**

As expected, the UML diagrams that are used most frequently during software maintenance are class diagrams, use case diagrams, sequence diagrams and activity diagrams.
UML diagrams are used most frequently by architects (54%), SW analysts (30%) and managers (39%). But 40% of programmers use UML diagrams for software maintenance tasks, implying that the investment made by architects when creating UML diagrams has a payback: improvement of maintainer understanding. We noted, however, that there are some (1%) maintainers who do not have diagrams available from the development phase. In other cases, there are some maintainers (23%) for whom diagrams are available, but never used. This study also revealed that the educational level seems to be an influential factor as regards the use or not of UML. Apparently, the higher the education level, the greater the use of UML diagrams.

Additionally, and in relation to UML documentation, a majority use UML diagrams that are human-based diagrams (81%). The rest use UML diagrams generated using RE with a tool: 3% use pure RE diagrams, while 19% use RE UML diagrams reviewed by humans.

The results of this survey might be beneficial for helping companies see how to invest in making the systems being maintained easier to understand. These results give us grounds to encourage software developers, albeit with caution, to develop UML diagrams in the early stages of software development. That would facilitate future maintenance tasks, encouraging maintainers to keep diagrams updated.

These findings are a first approach to discovering the contexts in which companies use UML during software maintenance; we plan to investigate this topic in greater depth. This future work might take the form of a survey that includes open questions or interviews. We also wish to extend our investigation to involve industries that do not belong to our industrial contact networks.

Acknowledgments: This work has been funded by the following projects: GEODAS-BC (Ministerio de Economía y Competitividad and Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01) and IMPACTUM (Consejería de Educación, Ciencia y Cultura de la Junta de Comunidades de Castilla La Mancha, y Fondo Europeo de Desarrollo Regional FEDER, PEII11-0330-4414).
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

ABSTRACT

Context: UML is a commonly-used graphical language for the modelling of software. Works regarding UML’s effectiveness have studied projects that develop software systems from scratch. Yet the maintenance of software consumes a large share of the overall time and effort required to develop software systems. In this study we, therefore, focus on the use of UML in software maintenance.

Objective: Our goal is to elicit the practices of the software modelling used during maintenance in industry. Moreover, we wish to understand what are perceived as hurdles and benefits when using modelling.

Method: In order to achieve a high level of realism, we performed a case study in the ICT department of a multinational company. The analysis is based on 31 interviews with employees who work on software maintenance projects. The interviewees played different roles and provided complementary views about the use, hurdles and benefits of software modelling and the use of UML.

Results: Our study uncovered a broad range of modelling-related practices, which are presented in a theoretical framework that illustrates how these practices are linked to the specific goals and context of software engineering projects. We present a list of recommended practices that contribute to the increased effectiveness of software modelling.

Conclusions: The use of software modelling notations (like UML), is considered beneficial for software maintenance, but needs to be tailored to its context. Various practices that contribute to the effective use of modelling are commonly overlooked. This suggests that a more conscious and holistic approach with which to integrate modelling practices into the overall software engineering approach is required.

Keywords: UML, Software Maintenance, Modelling Languages, Industrial Case Study

6.1. INTRODUCTION

Modelling is a common practice in software engineering, and UML (OMG, 2011) is the de-facto standard notation for this (Dobing and Parsons, 2006; Scanniello et al., 2010). However, in-depth evidence regarding the practice of using UML in industry is very scarce. Studies into the effectiveness of UML have looked mostly at projects that develop software systems from scratch. Yet the maintenance of software consumes a large share of the overall time and effort required to develop software systems. Unlike developers, software maintainers, spend a lot of their time understanding the system. This has led us to pay particular attention to the use of UML diagrams by the maintainers of software systems, which consequently motivated this study. We believe that industrial studies of this nature on software modelling could contribute to an understanding of how modelling can be used effectively. Empirical studies are necessary in real environments if we are to increase the depth of the knowledge and validity of the application of Software Engineering practices (Runeson and Höst, 2009).
A considerable amount of software development effort concerns the maintenance of software. Indeed, maintenance typically consumes between 40% to 80% of software project costs (Pressman, 2005). It is, therefore, important to understand the impact that software modelling may have on software maintenance. We are convinced that the way in which UML is used – i.e. which practices are used – is an important contextual factor that is associated with the effectiveness of the use of UML. In order to understand the effectiveness of UML, we have elicited factors related to the costs and benefits of using modelling during software maintenance. In doing so, we add fresh findings to the hitherto scarce evidence regarding the payoffs and costs of software modelling.

There is only scant empirical evidence related to our topic, the majority of which has been obtained in academic contexts (Dzidek et al., 2008; Fernández-Sáez et al., 2014, 2015a). However, it is important to investigate whether the results obtained in a controlled context are valid in an industrial one. A case study explores a phenomenon within its real context, especially when the boundaries between phenomenon and context are not evident (Yin, 2002). The essence of the case study method is to conduct an empirical inquiry within its real-life context and to thereby provide detailed, qualitatively rich, contextual description and analysis of a complex real-life phenomenon. The relevance of the findings should be sought in not only their novelty, but also in the confirmation of ‘known’ practices, as well as in the contextualisation of the object of study. We therefore believe that the case study approach is suitable for our study, because we wish to gather detailed information concerning an industrial software maintenance project and to obtain reflections from practitioners as regards the failures and success (especially in terms of costs and benefits) involved in the use of UML diagrams (or in their absence). In this paper, then, we present empirical findings obtained in the ICT (Information and Communication Technology) department of a large multinational company by means of case study research.

The principal goal of our research is to discover what practices industrial software professionals employ when using UML, and how they perceive the effectiveness of software modelling, paying particular attention to software maintenance tasks. As mentioned above, we focus our attention particularly on UML as a specific modelling language because it is widely used in industry (Dobing and Parsons, 2006; Scanniello et al., 2010).

Since the term ‘maintenance’ is very broad, we would like to limit its scope from the beginning of this paper. That being so, in this paper, “maintenance” always refers to adaptive, perfective and corrective maintenance (Pigoski, 2001), while preventive maintenance is out of the scope of this study.

This paper is organised as follows. Section 6.2 presents the related work. Section 6.3 describes the case study and its design. The results obtained are set out in Section 6.4, whilst some recommendations are provided in Section 6.5. In Section 6.6 a summary of all the results, organised by research question, is presented, while the
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

Threats to validity are discussed in Section 6.7. Finally, Section 6.8 outlines our main conclusions, and Section 6.9 presents future work.

6.2. RELATED WORK

In order to find the relevant work related to this paper, we performed a Systematic Mapping Study (SMS) to discover all the empirical studies related to the use of UML diagrams in software maintenance, and to the understandability and modifiability of UML diagrams as regards how they might influence system maintenance (Fernández-Sáez et al., 2013a). The findings obtained in this SMS are presented in the first block of related work. As the SMS was performed a few years ago, we have attempted to update its results to discover more recent evidence related to the topic discussed in this paper. In the second part of this section, we go on to present experiments or academic evidence related to the use of UML. The last block of this section is similar to the second, because it also complements the evidence found in the SMS, but it contains industrial rather than academic evidence.

The SMS (Fernández-Sáez et al., 2013a) enabled us to discover 46 papers reporting 74 empirical studies related to the topic of the use of UML diagrams in software maintenance. Of these, only the following experiments were directly related to the maintenance of source code:

- Dzidek et al. (Dzidek et al., 2008) performed a controlled experiment to investigate whether the use of UML influences maintenance, as compared to the use of only source code. This experiment investigates the costs of maintaining, along with the benefits of using, UML documentation during the maintenance and evolution of a real nontrivial system, with 20 professional developers as subjects. These maintainers had to perform five maintenance tasks consisting of adding new functionalities to an existing system, after which the correctness, time and quality of the solution were measured. Both the source code and the UML diagrams, when available, had to be maintained. The results of this work show a positive influence of the presence of UML for maintainers. UML was always beneficial in terms of functional correctness (with fewer faults being incorporated into the software) because the subjects in the UML group showed, on average, a practically and statistically significant 54 percent increase in the functional correctness of changes. UML also helped produce code of better quality when the developers were not yet familiar with the system. This experiment is a replication of previous work (Arisholm et al., 2006) performed with students, which obtained similar results.

- Arisholm et al. (Arisholm et al., 2006) presented the results of two controlled experiments carried out to assess the impact of UML design diagrams on software maintenance. 98 undergraduate students were involved. The authors analysed the time taken to perform the modifications to the system, the time spent on maintaining the models, and the quality of the modifications performed. The results of the quantitative analysis revealed no significant difference as regards the time spent making the modifications. Like (Dzidek et al., 2008), they observed that the quality of the modifications was higher for those participants who were equipped
with UML diagrams. As in (Dzidek et al., 2008), the participants’ ability and experience were not analysed with regard to the comprehensibility and modifiability of source code. Unlike our study, the authors analysed the effect of UML-based documentation (a use case diagram, sequence diagrams for each use case, and a class diagram) on modification tasks performed on both UML diagrams and source code.

In order to extend the results obtained in the aforementioned SMS, we would like to mention some other empirical studies, which were not part of the SMS, owing to their particular contexts or dates. We shall first briefly describe other recent experiments performed using undergraduate students, and which are related to UML and maintenance:

- A comparison of the attitude and performance of maintainers when using forward designed (FD) diagrams vs. reverse engineered (RE) diagrams during the maintenance of source code is discussed in (Fernández-Sáez et al., 2015a). The results show a tendency towards obtaining better results when using UML diagrams (class diagrams specifically) that were hand-made during the design phase. This is because the participants preferred FD diagrams when understanding and maintaining a system, although their performance was not, in some cases, much better with FD diagrams. They also noted that those participants who received RE diagrams had more difficulties when reading the diagrams, especially the sequence diagrams.

- Focusing on the possible advantages of Model-Driven Development (MDD), improvement to maintainability is studied in an experiment presented in (Ricca et al., 2012). The results, which were obtained with Unimodal (a specific implementation of executable UML), indicate a relevant shortening of time with no significant impact on correctness, in comparison to conventional manual programming when performing maintenance tasks.

- In (Scanniello et al., 2012) an experiment is presented, whose aim was to assess whether the comprehension of source code is influenced by the presence or otherwise of UML class and sequence diagrams produced in the design phase. The results reveal that the availability of UML allowed the subjects to perform the maintenance tasks better.

- The results of a family of 4 controlled experiments presented in (Scanniello et al., 2014) reveal that the use of analysis-level UML diagrams does not significantly improve the comprehension and modifiability of source code with regard to the use of source code alone.

- The experiment in (Fernández-Sáez et al., 2014) studies whether different Levels of Detail (LoD) in UML diagrams influence the maintenance of source code. The results suggest that high LoD diagrams are more helpful when understanding a system in comparison to low LoD, while low LoD diagrams are more helpful when carrying out maintenance tasks.
• Leotta et al. (Leotta et al., 2013) conducted a pilot experiment with 21 Bachelor’s degree students, aiming to investigate the effect of documentation accuracy during software maintenance and evolution activities. The result they obtained was a benefit of 15% in terms of efficiency when more accurate documentation was used. Part of their experiment revealed that UML documentation is considered to be more accurate.

The pattern that emerges from the results of these experiments, under controlled conditions in academic environments, is that the use of UML diagrams is, to some extent, beneficial for software maintenance. One important issue was to study whether these results also hold in an industrial environment. The industrial evidence related to the topic of this paper is the following:

• Scaniello et al. (Scaniello et al., 2010) presented the results of an exploratory survey used to investigate the state of practice regarding the use of UML in software development and maintenance. The majority of the companies interviewed (75% out of 22 companies) used UML for software development and maintenance. The interviewees were mainly practitioners with little experience as regards performing maintenance operations. Another interesting point concerns the average effort needed to perform maintenance operations, which ranges from 1 to 5 person hours for an ordinary maintenance operation (e.g., corrective changes), and from 10 to 50 person hours in the case of an extraordinary maintenance operation (e.g., perfective or adaptive changes). Scaniello’s study provides the responses of 22 Italian companies, which means that the study cannot be generalised to other companies throughout the world.

• Several related pieces of work investigate aspects of software modelling in general and/or model-driven approaches (MDA). If we focus on those surveys directly related to the influence of software modelling with software development, the work of Anda et al. (Anda et al., 2006) should be highlighted. The paper in question reported the anecdotal advantages of modelling, such as improved traceability, but also pointed to potential negatives, such as increased time taken to integrate legacy code with models, and organisational changes needed to accommodate modelling.

• There is also a paper based on the study of Model Driven Engineering (MDE) from an empirical point of view (Hutchinson et al., 2014). The authors first present the results of a survey of MDE deployment, and provide some rough quantitative measures of MDE practices in industry. They then go on to supplement these figures with qualitative data obtained from some semi-structured, in-depth interviews with MDE practitioners In particular, they supplement their figures by adding a description of the practices of four commercial organisations as they adopted a MDE approach for their software development practices. In documenting some details of their attempts to deploy model driven practices, the authors identify a number of contextual factors, in particular the importance of complex organisational, managerial, and social factors (as a complement to technical factors) that appear to influence the relative success, or failure, of the endeavour.
The interviewees describe genuine success in their use of model driven development, but explain this as examples of organisational change management. They conclude that the successful deployment of MDE appears to require a progressive and iterative software development approach, transparent organisational commitment and motivation, integration with existing organisational processes, and a clear business focus.

All the related work is summarised in Table 6.1, thus providing the reader with an overview of the main information with which to compare the empirical studies. The columns in the table have the following content:

- **Ref**: contains the reference to the paper that presents the empirical study considered.
- **Type of empirical study**: indicates the type of empirical study summarised in the paper (a survey, an experiment, a family of experiments, etc.).
- **Goal**: describes the goal pursued by the empirical study.
- **Subjects**: presents the numbers of subjects who participated in the empirical studies, as well as the type of subjects (students, professionals, academic staff, etc.).
- **Independent variables**: describes the variables that are studied, to ascertain their effect on the dependent variables. The values (treatments) of the independent variables are also presented.
- **Dependent variables**: presents the outcome variables, which are the variables that are affected by the changes produced in the independent variables.
- **Experiment design**: contains the type of design selected, which can be between-subjects (each subject receives only one treatment) or within subjects (each subject receives all the treatments).
- **Tasks**: describes the tasks to be performed by the subjects as part of the empirical study.
- **Results**: reveals the main findings obtained.

### 6.3. RESEARCH METHOD

As mentioned above, we selected the case study as the research method that would be applied, in order to obtain empirical evidence from a real environment. We believe the case study approach is suitable for our study because we wish to gather detailed information about industrial software maintenance projects and to obtain reflections from practitioners on the failures and success in relation to the use of UML diagrams (or their absence).

In this section, we discuss aspects of the research method employed in our study, following the guidelines for case studies proposed by (Cruzes et al., 2011; Höst and Runeson, 2007; Runeson and Höst, 2009; Runeson et al., 2012).
**Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?**

**Table 6.1. Summary of related work (part 1/3).**

<table>
<thead>
<tr>
<th>Ref</th>
<th>(Dzidek et al., 2008)</th>
<th>(Arisholm et al., 2006)</th>
<th>(Fernández-Sáez et al., 2015a)</th>
<th>(Ricca et al., 2012)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type of study</strong></td>
<td>1 experiment</td>
<td>2 experiments</td>
<td>Family of experiments (1+2)</td>
<td>1 experiment</td>
</tr>
<tr>
<td><strong>Goal</strong></td>
<td>To investigate whether the use of UML influences maintenance in comparison to the use of only source code.</td>
<td>To investigate whether the use of UML influences maintenance, in comparison to the use of only source code.</td>
<td>To determine the influence of the origin of UML diagrams on source code maintenance.</td>
<td>To evaluate the effectiveness of Model-driven development (using UniMod) during software maintenance tasks.</td>
</tr>
<tr>
<td><strong>Subjects</strong></td>
<td>20 professional developers.</td>
<td>Undergraduate students (22 and 76, respectively).</td>
<td>Undergraduate students (40, 51 and 78, respectively).</td>
<td>21 Bachelor’s degree students.</td>
</tr>
<tr>
<td><strong>Independent variables</strong></td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The use of UML documentation in a UML-supported IDE (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The origin of UML diagrams (possible values: forward design or reverse engineered diagrams).</td>
<td>Effectiveness of UniMod vs. Java.</td>
</tr>
</tbody>
</table>
| **Dependent variables** | - Time needed to change source code.  
- Time needed to change source code + UML diagrams.  
- Functional correctness of the solution. | - Time needed to change source code.  
- Time needed to change source code + UML diagrams.  
- Functional correctness and quality of the solution.  
- Quality of the change. | - Understandability of source code.  
- Modifiability of source code. | - Time required to perform maintenance tasks.  
- Artefact correctness. |
| **Tasks** | Modification tasks in source code and in UML diagrams. | Modification tasks in source code and in UML diagrams. | Modification tasks + Subjective questions. | Maintenance tasks in source code and in UniMod. |
| **Results** | The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system. | The UML subjects took more time if the UML documentation was to be updated. UML was always beneficial in terms of functional correctness. UML also helped produce better quality code when the developers were not yet familiar with the system. | Tendency to obtain better results when using class diagrams, which were hand-made during the design phase based on statistical and subjective results. Reversed-engineered sequence diagrams were considered difficult to read. | Results indicate a relevant shortening of time with no significant impact on correctness, gained through the use of UniMod instead of conventional programming (i.e. code centric programming). |
Table 6.1. Summary of related work (part 2/3).

<table>
<thead>
<tr>
<th>Ref</th>
<th>(Scanniello et al., 2012)</th>
<th>(Scanniello et al., 2014)</th>
<th>(Fernández-Sáez et al., 2014)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of study</td>
<td>1 experiment</td>
<td>Family of experiments (1+3)</td>
<td>Family of experiments (1+3)</td>
</tr>
<tr>
<td>Goal</td>
<td>To investigate whether the comprehension of source code increases when participants are provided with UML class and sequence diagrams produced in the software design phase.</td>
<td>To investigate whether the use of UML models produced in the requirements analysis process helps in the comprehensibility and modifiability of source code.</td>
<td>To determine the influence of different levels of detail (LoD) of UML diagrams in source code maintenance.</td>
</tr>
<tr>
<td>Subjects</td>
<td>16 undergraduate students.</td>
<td>Undergraduate students (24, 22, 22 and 18, respectively).</td>
<td>Undergraduate students (11, 16, 32 and 22, respectively).</td>
</tr>
<tr>
<td>Independent variables</td>
<td>The use of sequence and class diagrams created in the design phase (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The use of UML analysis models (possible values: presence or absence of UML diagrams accompanying source code).</td>
<td>The level of detail of UML diagrams (possible values: high or low LoD).</td>
</tr>
<tr>
<td>Dependent variables</td>
<td>Comprehension of source code.</td>
<td>-Comprehension level of the source code. -Capability of a maintainer to modify source code.</td>
<td>Understandability and modifiability of source code.</td>
</tr>
<tr>
<td>Experiment design</td>
<td>Comprehension questions.</td>
<td>Comprehension and modification tasks and subjective questions.</td>
<td>Comprehension tasks + modification tasks + subjective questions.</td>
</tr>
<tr>
<td>Tasks</td>
<td>Within-subjects.</td>
<td>Within participants Counter-balanced design.</td>
<td>Between-subjects.</td>
</tr>
<tr>
<td>Results</td>
<td>Participants comprehend source code significantly better when class and sequence diagrams are added together.</td>
<td>UML models produced in the requirements analysis process influence neither the comprehensibility of source code nor its modifiability.</td>
<td>The descriptive statistics show that high LoD diagrams are more helpful when understanding a system in comparison to low LoD, while low LoD diagrams are more helpful when carrying out maintenance tasks.</td>
</tr>
</tbody>
</table>
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

Table 6.1. Summary of related work (part 3/3).

<table>
<thead>
<tr>
<th>Ref</th>
<th>Type of study</th>
<th>Goal</th>
<th>Subjects</th>
<th>Independent variables</th>
<th>Dependent variables</th>
<th>Experiment design</th>
<th>Tasks</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Leotta et al., 2013)</td>
<td>1 survey</td>
<td>To investigate the impact of non-aligned UML documentation on maintenance tasks.</td>
<td>21 third year Bachelor’s degree students.</td>
<td>The alignment of class diagram and source code (Possible values: &quot;less&quot;/&quot;more&quot; aligned documentation)</td>
<td>Efficiency</td>
<td>Counter-balanced design.</td>
<td></td>
<td>The results confirmed that an aligned documentation helps during maintenance tasks (15% benefit).</td>
</tr>
<tr>
<td>(Scanniello et al., 2010)</td>
<td>1 case study</td>
<td>To investigate the state of the practice regarding the use of UML in software development and maintenance in Italian industry.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Anda et al., 2006)</td>
<td>1 survey + interviews</td>
<td>To identify immediate benefits, along with difficulties and their causes, when introducing UML-based development into large projects.</td>
<td>16 system developers and project managers.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Hutchinson et al., 2014)</td>
<td>-</td>
<td>To describe and understand the industrial experience of MDE.</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The majority of the companies interviewed use UML for software development and maintenance. Practitioners with little experience perform mainly maintenance operations.

The interviewees made improvements to traceability from requirements to code, design of the code, and development of test cases, and also to communication and documentation.

Some factors that appear to influence the success/failure of applying MDE approaches, as opposed to simple technical factors, were identified: in particular, the importance of complex organizational, managerial and social factors.
6.3.1. Goal and research questions

In the introduction, we affirmed that the principal goal of our research is to discover what practices industrial software professionals use when using UML, as well as to find out how they perceive the effectiveness of software modelling, paying particular attention to software maintenance tasks.

The use of the Goal-Question-Metrics template (Basili and Weiss, 1984) enabled us to formulate the goal of this case study as follows: “Analyse the use of software modelling for the purpose of investigating its use, effectiveness and hurdles, with regard to software maintenance tasks, from the perspective of the researcher, in the context of an industrial ICT department”.

This goal was used to formulate our main research questions (RQs):

- **RQ1) What practices are involved in using UML in software maintenance projects?**
  This research question was directly derived from the GQM statement. We wanted to understand the impact of UML, and suspected that different people used UML in different ways. Through this question, we wanted to elicit the variety of practices, because different practices are likely to have different kinds of impact on effectiveness of use of UML.

- **RQ2) What are the costs-factors and benefit-factors of using UML in software maintenance projects?**
  This research question arose from RQ1, because costs and benefits are drivers that shape the practices of using UML. This RQ thus needs to be answered in an effort to help find a response for RQ1.

Although the main objective of this study is to attempt to answer the RQs stated above, two additional RQs were also considered when designing and analysing this case study. The following RQs were added, so as to provide a comparison between UML documentation, and documentation that does not contain diagrams (UML diagrams in particular, and other modelling notations in general):

- **RQ3) What are the factual and perceived hurdles when maintaining documentation, and UML models as part of that documentation?**
  Apart from providing a comparison of documentation containing models vs. documentation which does not contain any kind of models, this RQ separates the subjective point of view concerning the obstacles to the correct maintenance of the documentation of a project found by maintainers from the factual, objective perspective.

- **RQ4) What are best practices when using diagramming and modelling in documentation?**
  After eliciting and reflecting on the costs and benefits, as well as the hurdles experienced by maintainers, we try to distil what can be considered the best practices, and go on to provide some recommendations about (UML) modelling documentation, and SE practices in general.
6.3.2. Case and subject selection

For our case study, we collected data at the ICT Department of a multinational transport company in Western Europe. This company is about 100 years old, but we could trace the use of ICT in the company back to (at least) the late 1960’s. The ICT department has between 800-1000 employees, all of whom are involved in ICT functions. ICT plays an essential role in this company’s competitive advantage within its sector. Moreover, the company has to comply with many ICT standards in the sector. An internal department produces the company’s software in-house, and its innovation as regards ICT is a valuable asset. The organisational unit of interest for this research is located within the Information Services development department, which is part of the ICT department. The Information Services development department is where all the software development takes place.

The types of software systems that are being maintained at the company are the following:

- **Mainframe systems**: these started in the late 60’s, signifying that low level programming languages are used. Programmers involved in the maintenance of this kind of systems are hired after completing their school studies, and trained in-house. This training is based on high-ceremony and documentation-intensive development processes.

- **Information systems**: this kind of systems includes several hundreds of applications, several of which are considered to be legacy systems. They usually contain a fair number of interfaces to external systems. Employees with heterogeneous backgrounds perform the maintenance of this kind of systems.

- **Mobile systems**: these systems typically use web and agile technologies. Younger developers who have had some form of UML training as part of their formal education are usually involved in the maintenance of this kind of systems.

The ICT employees work in different divisions. These divisions are organised by domain-area. Another organisational split is in business technology. People in the technology domain are grouped in technology-areas: mainframe, mobile and web, Java, SAP, etc. The ICT organisation can therefore be considered as a matrix structure (Figure 6.1). The functions that people have in a team may additionally be specialised in a particular activity of the software development process: architect, developer, tester, tool-smith, business-analyst, information-analyst, project manager, etc.

In this organisation, all software is developed through projects, which have a limited duration in time. When a new system is provided, the Company call that project a “development project”, while an existing system that is modified or adapted is called a “maintenance project” (that is, large maintenance changes are made in the form of new development projects). The creation and update of documentation is part of the responsibility of the projects themselves. Software maintenance is carried out by a specialised and dedicated team in the organisation. Most of the projects in this ICT department are in fact mainly software maintenance in type. We shall therefore
provide a detailed definition of maintenance (which is also the explanation we gave to all the participants in the case study).

By “software maintenance” we refer firstly to small changes made in a system that is running, and changes that are intended to remove errors or bugs from the software, the procedures, the hardware, the network, the data structures, and the documentation”, i.e. corrective maintenance (Swanson, 1976). We also refer to major system changes like maintenance activities intended to enhance the system by adding features, capabilities, and functions, in response to new technology, upgrades, new requirements, or new problems, i.e. a modification of a software product performed after delivery to keep a software product usable in a changed or changing environment (ISO/IEC, 1999). This might be regarded as adaptive and perfective maintenance. In this study, preventive maintenance is not taken into account. There is a company-wide standard for software development. This standard prescribes the development process and deliverables, milestones, approval procedures, and quality assurance (which includes naming conventions for source code). This standard has grown out of an iterative development style. The company is also adopting agile development for some of its smaller projects, and would like to scale this up to larger projects. The standard does not dictate the use of UML or modelling in general. Enterprise-wide integration across systems is based on the service orientation (SOA) paradigm.

The tools available for modelling at the company are: Visio, Bizz Design Architect and Sparx Enterprise Architect. The tools for source code management commonly used in the software development/maintenance projects in this company are:

- JIRA: this is an issue tracking system used for bug tracking and issue tracking.
- Subversion: this is used for version management of source code.
- Tools for code coverage and code quality are also in place.

We would also like to summarise the types of documents that are commonly used in a typical maintenance project at this company. In addition, we have related each specific responsibility role to a document. This information is summarised in Figure 6.2.
Figure 6.2. Flow of documentation in the company and responsibility roles.
Our study is a single embedded case study (Figure 6.3) and follows the classification by Yin (Yin, 2002). Our unit of analysis is the “approach to modelling”. For this purpose we studied various projects within one large IT-development department (cases). In studying our cases, we quickly found out that to understand the approach to modelling, we also needed to understand the context of these projects. In particular, we had to comprehend the goals of the various stakeholders in the software development organization. Within these cases, then, we looked at modelling goals, processes, practices and tools (as the main aspects of the ‘approach to modelling’) from the perspectives of different stakeholders in those projects.

We chose this company because one of the authors of this paper had arranged placements for MSc. students at that organisation, which meant that there was a collaboration relationship. The company is considered to be a fair representative of large multinational companies, having a large ICT department, a heterogeneous IT-landscape, and a long record of accomplishment in software development.

6.3.3. Data collection procedures

The source used to obtain data regarding the use of UML during maintenance tasks was interviews with Company personnel. In order to avoid trouble related to ethical issues, there was no information in the raw data that could allow a particular individual to be identified. A note was made of the names of the interviewees who participated in the case study, so as to be able to contact them when the interviews were transcribed and obtain written confirmation of their agreement to the interviews being used, but it was not possible to link their names to their responses. We used semi-structured interviews in which the interviews are “guided conversations” (McNamara, 1999). The interviews were standardised, in the sense that each interviewee was asked similar questions, but they were also open-ended, in that there was ample room for the interviewees to elaborate. We must also take into account that no choices were provided to interviewees for each question. This means that all the results of this case study are spontaneous responses on the part of the interviewees. The interview questions are shown in APPENDIX E. INTERVIEW QUESTIONNAIRE (related to Chapter 6) The 33 questions (21 main questions, some of them with subquestions) were selected using a refinement and adaptation of questions from previous empirical studies as our basis. We started with the
questionnaire of a previous survey done by the authors of the paper (Fernández-Sáez et al., 2015b). We then complemented the list of questions using as inspiration other empirical studies related to software maintenance (de Souza et al., 2005; Yamashita and Moonen, 2012). Apart from those sources, studies from different contexts, such as Embedded Software Engineering or MDE, were taken into account, but these were (partially) focused on UML (Hutchinson et al., 2014; Mellegård and Staron, 2010; Torchiano et al., 2013).

6.3.4. Case study execution and analysis procedure

The first author spent 12 months as a temporary member of the organisation in the capacity of research intern. She had direct access to the company staff, and in particular to the people involved with the maintenance projects. We first collected the data by performing 37 interviews of about one hour each. This evidence was gathered over the period 2012 - 2014. Six of the interviews were discarded because the employees were not related to software maintenance issues. In the end, 31 interviews were completely analysed, and are reported in this paper.

We started with a list of questions, because we were very interested in learning about those topics. At the same time, the interviews were done in an open form: certain questions were asked to some interviewees, and different ones were given to others. In addition, some of the questions were focussed more on one particular issue (e.g. about UML) and others were about more general topics, such as documentation in general. It should also be said that a number of insights were found that are not related to UML (the main factor under study in this case). These insights came to light during the interviews, and we allowed the interviewees to continue talking about these topics - in the spirit of grounded theory. During the research period, interviews that had already been done were also coded, in order to guide the subsequent interviews. The list of questions was used especially as support in starting a new conversation when a deadlock was reached in the existing conversation. That happened because sometimes interviewees were "shy" and did not talk too much (especially about those topics in which they did not consider themselves experts). We would like to stress that all the questions were open-questions. This meant that no choices were presented to the interviewees; the findings presented in Section 6.4 are a result of a process of:

- coding the interviewees’ spontaneous responses,
- grouping the codes obtained,
- analysing the groups obtained.

The interviews were performed with people playing different roles, so as to obtain different points of view. The interviewee roles include: project manager, information analyst, project architect, technical lead, programmer or application developer, test engineer, delivery lead, SCRUM master, and systems analyst.

In order to extract findings from our study, we used the grounded theory approach, which is built upon two key concepts: constant comparison, in which data are
collected and analysed simultaneously, and theoretical sampling, in which decisions about which data should be collected next are determined by the theory that is being constructed (Glaser and Strauss, 1967). The distinctive feature of a grounded theory approach is its commitment to research and “discovery” through direct contact with the social world of interest, coupled with a rejection of a priori theorising (Locke, 2001).

The essential idea in discovering a grounded theory is to find a core category, at a high level of abstraction but grounded in the data, which accounts for what is central in the data (Punch, 2005). This is done in the following three stages (Robson, 2002):

- Finding conceptual categories in the data (coding).
- Finding relationships between these categories.
- Conceptualising and accounting for these relationships by finding a core category.

The analyst begins the conceptualisation with open coding, which is “the part of analysis that pertains specifically to the naming and categorising of phenomena through the close examination of data” (Strauss and Corbin, 1990). This is accomplished largely by asking questions about data and making comparisons, aiming to find similarities and differences between each incident, event, and other instances of phenomena (Strauss and Corbin, 1990).

Following this approach, and using the qualitative data, i.e. the data obtained from the interviews, we analysed the data in the following steps (Figure 6.4):

1. We performed the interviews, and the conversations were recorded using a voice recorder.
2. Each interview was transcribed. We used the Digital Voice Editor tool (Sony, 2010) and a text processor (Word) in order to perform this process. The transcriptions of the interviews were shown to the respective interviewees, and they either accepted them, or clarified what they had intended to say.
3. We analysed each transcription, highlighting the important and surprising statements. This was done through the simultaneous use of NVivo 10 (Richards, 1999) and Word by means of open thematic coding. This coding was carried out independently by two researchers, and then discussed.
4. We then coded the statements and grouped them under more general categories or factors (Seaman, 1999). We also used NVivo 10 and Word to perform this step.

The transcribed interviews led to a set of text files of 146,821 words in total. The quantitative data was derived from the background questions that we asked the interviewees.

Finally, we summarised all the results of this case study, generating a theory. In mature sciences, building theories is the principal method of acquiring and accumulating knowledge. But there is little use and development of empirically-based theories in Software Engineering (Sjøberg et al., 2008). Theory is the means by which one may generalize analytically (Cook et al., 2001), thus enabling generalization from situations, such as case studies, in which statistical generalization is not desirable or
possible (Yin, 2002). In case-based research, we may attempt generalization from the object of study to the theoretical population immediately. For example, from the investigation of a single project, we may tentatively hypothesize a generalization about all similar software engineering projects in similar companies. Another aspect of case-based research is that variability is reduced by the breaking down of a single case into components with interactions, such as for example people and roles in a project. These components and mechanisms may be recurrent across a large set of different cases, and are hence interesting subjects of generalization. It is important to note that since a conceptual framework is a set of definitions, it cannot be true or false, but it can be applicable or not (Wieringa and Daneva, 2015).

We decided to generate a theory as the final result of the grounded-theory process, motivated especially by the comment of Johnson et al.: “many proposed [...] methods, programming languages and requirements specification languages exist, but very few explicit theories explain why or predict that one method or language would be preferable to another under given conditions” (Johnson et al., 2012). It is not worthwhile to develop a general theory of software engineering, but it is very useful to develop incompletely specified, partial theories that can be applied to practice (Wieringa and Daneva, 2015).

6.4. RESULTS

This section presents the results of our case study. The main results of this empirical study are qualitative data, although in some cases we have provided a few
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

Statistics when these help to reinforce the importance (by repetition) of a specific finding. In order to present all of them in the best way, organizing them and linking them when possible, we have produced a theory inspired by the elements highlighted in (Sjøberg et al., 2008), along with the authors’ (previous and current) observations of industry practice. Our theory is formulated on the basis of conclusions drawn and extracted from observations in industry, the objective being to capture all the observations of this particular case study and help explain them.

The diagrams of our theory contain the constructs (what the basic elements are), and propositions (how the constructs interact). The explanations (why the propositions are as specified) of the theory are detailed in the text accompanying the diagrams. Note that the scope (what the universe of discourse is in which the theory is applicable) of the theory is the same as the context of the case study. We used the nomenclature suggested in (Sjøberg et al., 2008).

The theory is presented by means of multiple diagrams that focus on different areas of the theory. Figure 6.5 presents the “baseline” theory. On it we represent the environment in which the results of the case study will be embedded. On the left side, we can observe elements related to a typical SE approach. We selected only the main elements that we found in our case study, but there are definitely other elements which might be good candidates for being represented also there. On the right side we represent the approaches of interest in this case study: the approach to modelling (and especially the approach to UML modelling), and also the approach to documentation (for example when modelling is not available).

The main statements represented in this theory are the following:

- Projects take place in a context, have stakeholders and can be in a particular stage of a lifecycle.
- Stakeholders have goals, and these may change throughout the execution stages of a project.
- The stakeholders’ goals drive the SE process used and the practices used in the overall approach to SE.
- A process denotes the collection of (formalised) steps of tasks that the project follows to engineer software.
- The processes and practices in turn drive the choice and use of tools.
- Part of the overall approach to SE is the approach to documentation (AtD) and the approach to implementation (AtI). AtD and AtI refer to a combination of goals, processes, practices and tools for documentation and implementation, respectively.
- Together, the AtD and AtI drive the approach to modelling (AtM). The approach to modelling itself again consists of a goal-modelling process, a set of modelling practices and a collection of modelling tools. The modelling approach (which is based on UML or another notation) represents a “bridge” between the AtD and the AtI.
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

We would like to point out that the boxes containing others do not necessarily mean an inheritance relationship; a “part of” relationship is more appropriate in this case. The legend for all the diagrams that present our theory is summarised in Figure 6.6.

Figure 6.5. "Baseline" theory.

Figure 6.6. Legend of the theory-diagrams.

We present the results of the case study, grouped by each element of the SE approach represented in Figure 6.5, and we also made an effort to relate them to each RQ. As mentioned above, all the results in the following subsections were extracted using the grounded theory methodology.
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

Furthermore, the results of this case study are complemented with quotes from the interviews that were considered noteworthy as regards the topic being analysed. Those quotes are labelled using the term “Int”, followed by the number of the interviewee (for example, [Int12] would be used to label a quote from interviewee number 12). A summary of the main background information related to each interviewee is presented in APPENDIX F. We ask the reader to take into account that a few of the interviewees did not want to provide some of their personal details (educational level or field of education). Although it was emphasised to them that the interviewees were going to have their identity kept secret, these individuals thought that their responses might make it possible to link to them to those more personal data, so they preferred not to provide the personal details mentioned. It is also important to point out that since not all the interviewees responded to all the questions, the percentages shown in this paper were calculated in two ways:

- Relative to the total number of interviewees.
- Relative to the number of interviewees that mention this term/category.

The means used to calculate the numbers is stipulated in each section.

6.4.1. Background

We asked the interviewees to fill in a short questionnaire about their background in order to characterise them, but not all of them wished to give this information.

With regard to the interviewees’ gender, 10% of them are female and 90% are male. This proportion was expected, based on our personal perception of the proportions of ICT employees at the company.

The interviewees’ educational level would appear to be mostly (11) Bachelor Students, e.g. polytechnic level, and Master’s Students, at universities (4). The fields of education they come from are mainly from Computer Science degrees, although there are other fields, such as Arts, Business and Finance, Chemistry and Physics, Electronics, Maths, Psychology and the Navy.

We also asked our interviewees about their experience in ICT. We classified them into the following categories:

- **Low experience**: those who have been working in ICT for less than one year (2 interviewees).
- **Medium experience**: those who have been working in ICT for between one and five years (2 interviewees).
- **High experience**: those who have been working in ICT for between five years to 10 (4 interviewees).
- **Very high experience**: those who have been working in ICT for more than 10 years (19 interviewees).

The majority of the interviewees are very highly-experienced people.

The roles of the interviewees were varied: analyst developer (1), delivery lead (1), deployer (1), information analyst (3), program analyst (1), programmer / application
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

Half of our interviewees (16) responded to the question about the projects on which they were working at the time of the interviews (the others preferred not to say, as they were concerned about privacy). The types of projects were varied:

- Common projects, concerning desktop or web applications (31%)
- Mobile application projects (25%)
- Projects regarding the maintenance of old legacy systems (13%)
- Outsourced/offshored projects (13%)
- Projects concerning mainframe systems (6%)
- Projects regarding migration (6%)
- Projects related to embedded real-time programming (6%)

6.4.2. Goal

In this subsection, we present the results of our case study that are related to the element “GOAL.” Firstly, we focus on the different goals or purposes of using UML, all of them mentioned by the interviewees. This subsection contributes to answering RQ1. We also present a subsection about “costs,” which is a specific goal (to reduce the project costs), and common to the majority of maintenance projects. That subsection contributes to answering RQ2.

6.4.2.1. Purpose of use of UML

One of the questions during the interview was: “Why do you use UML diagrams? / For what purpose is UML modelling used?” The answers to these questions varied, and are shown in Figure 6.7. When reading the figure, please bear in mind that one interviewee might have mentioned more than one purpose, so it is important to note that the percentages represent the ratio of interviewees who spontaneously mentioned a purpose (they do not add up to 100).

The majority of the interviewees use UML as a communication tool (63%), in line with other empirical studies (Petre, 2013). This communication can be:

- Between team members (38%).
- With members of other teams (9%). This is especially important when part of a team is geo-distributed, or when part of the maintenance is outsourced. In these cases, the use of prototypes would also be very helpful as regards properly communicating what should be done.
- With stakeholders (6%). This type of communication is usually difficult, and UML helps to establish a standardised means of explaining the size and the complexity of the project to stakeholders.
- UML is also used to communicate the current situation to newcomers to the project (16%). This is very important in the quest to avoid “knowledge evaporation”. That
situation occurs when an expert on a system leaves a project, taking with him/her some knowledge that is not documented.

The wide use of UML as a representation for communication might be thanks to the fact that it is a standard notation, and because it is well known both by professionals and recent graduates. People also recognise that UML diagrams are used to complement verbal communication (face-to-face or written), but not to replace it: “UML helps to improve the communication, but it doesn’t replace it” [Int4].

The next most common use of UML diagrams is to obtain an overview of the system being maintained, or to summarise it (41%): “UML is mostly a high level picture or presentation of what the landscape looks like; so, by that I mean, this is a product which is placed in this domain; it uses these services, these data models, and they interact with each other in this way” [Int6].

The next most common use of UML is to help to create a proper design (22%): “UML is a sort of a tool you can use while thinking about and creating things; going over them again, you’ll find mistakes in the next step; you can adjust new elements, new classes and...so it’s also a sort of brainstorming that in that way supports, really supports, the specification process” [Int35]. With regard to sequence diagrams, they are used to help plan for a solution, but the diagrams are not created for documentation purposes: “We have often seen that developers start coding without really thinking through the programme that they have. [...] However, having a sequence diagram in place will help them to think through the problem before they solve it. [...] Sometimes, depending on the need, it may be kept or discarded, because when you cannot maintain it you discard it, but the purpose is actually fulfilled” [Int24].
Another purpose of using UML which is often reported as much as the previous one (22%), is to enhance people’s own understanding of the system being maintained. A modelling notation is also a tool which can help maintainers to create their own mental representations of the system being maintained, as well as of the task that needs to be carried out: “When you pick up something new, big, and challenging, then you start writing your thoughts on the whiteboard, and at some point on the whiteboard, the clouds converge to show something that is structured. So you make a diagram of it. You think it out. Then you can you back to the whiteboard and do the next level of understanding. And that is what is great about UML. [...] You can use the diagram easily to explain what you have in mind. That is what I usually do” [Int4].

When comparing these findings to related work, a question arose: how closely do these internal structures correspond to external representations, whether these are in relation to program code or to visualisation tools? This question was studied by Petre’s work (Petre and Blackwell, 1999), but her study is more oriented to the designing of program codes, rather than to architectural design. In our interviews we find that UML is used both by architects and programmers. Petre reports that software engineers use 2D, multidimensional and multi-model mental representations.

The next most cited reason for using UML was to guide testing and to plan the rest of the project, creating a to-do list (13%). As an illustration, we quote: “Sometimes I put in things just to remember that these are aspects that are also there, or [...] I put in the components that do not exist or seldom exist, just to remember <I have to do something with this> [...], then at least we don’t forget to address that aspect at the very beginning” [Int3]. Some project managers also use UML diagrams to keep track of progress [Int37].

Uses that were mentioned, but only rarely (3%; i.e. by only one person for each use), include: guiding implementation, analysing risks, documenting, following the mandatory process, justifying costs, supporting maintenance, determining responsibilities for success (offshore team), monitoring implementation, having a professional way of developing, checking the quality of the implementation, or showing progress. This list of rarely mentioned purposes of the use of modelling is aligned with the results of the survey by Liebel et al. (Liebel et al., 2016). The study in question focused on embedded software systems and discovered that the main purposes of using models in that domain are: simulation, code generation, and documentation. This makes sense because the automation of activities in the embedded domain during the development process would appear to be of great importance.

We should also comment that some possible purposes which we expected were not mentioned by any of the interviewees; these included purposes such as certification, deployment, generation of implementation, knowledge transfer or reasoning about design.
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

The responses to this question show that – when available - UML models serve a variety of purposes. The main purposes are rather ‘soft/fuzzy’ and are related to communicating, understanding and creating a design. Figure 6.8 shows the main purposes of the use of UML, along with the elements that influence it. In this figure, and in the other figures concerning the theory, the dashed lines represent evidence-based relationships. These relationships are represented using arrows, which link an element A with an element B. For example, the “influences” arrows mean that a decision in A has an impact on decisions in B. The “like” arrows mean that B is an example of A. The remaining names given to the arrows could be considered as self-explanatory (for example, “increases”, “leads to”, etc.).

6.4.2.2. **Perceived cost factors of modelling**

We asked the interviewees about the possible cost factors or investments related to the use of a modelling notation like UML in software maintenance: “*What cost factors are related to the use of UML modelling in your work?*”. Table 6.2 shows the responses to this question, and their ratios; these were spontaneous responses, i.e. the costs mentioned in this section were not suggested to the interviewees at any moment. The majority of the interviewees considered that training is an important investment.

The types of training costs mentioned could be split into two main groups. One significant factor that emerged was the cost of training in the UML notation. So we can say that in some sense the educational background may influence the discussion on the training cost of using models. This might be due to a fear of their own poor understanding of UML: “*They assume they understand the entire diagram, but the fact is that they completely misunderstand the diagram. So it depends on how you do the communication*” [Int3]. During the interviews we detected that the term UML is
sometimes considered to be a synonym of Rational Unified Process (Jacobson et al., 1999) or even Object Orientation (OO) (Blaha and Rumbaugh, 2004; Bruegge and Dutoit, 2010). Some of the interviewees thought that the use of a standard notation is not sufficient, and that training is necessary to establish alignments and conventions. It is also noteworthy that several engineers (especially those that have not passed through education in Computer Science or Software Engineering from a university), stated that they learn by doing, and that they train on the job. However, if they learn by looking at existing documentation, they will learn ‘box-and-line’ diagramming, with very little knowledge of the variety of syntactical elements, or of their official meaning.

Table 6.2. Cost factors related to the use of UML.

<table>
<thead>
<tr>
<th>Cost factor</th>
<th>% of interviewees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td></td>
</tr>
<tr>
<td>in UML notation</td>
<td>48%</td>
</tr>
<tr>
<td>in modelling tool</td>
<td>32%</td>
</tr>
<tr>
<td></td>
<td>16%</td>
</tr>
<tr>
<td>Tooling</td>
<td>32%</td>
</tr>
<tr>
<td>Migration of documentation</td>
<td>10%</td>
</tr>
<tr>
<td>Change of people’s mind</td>
<td>10%</td>
</tr>
<tr>
<td>Change of process</td>
<td>6%</td>
</tr>
<tr>
<td>Cost of not updating</td>
<td>6%</td>
</tr>
<tr>
<td>Cost of creation and updating</td>
<td>6%</td>
</tr>
<tr>
<td>Central governance</td>
<td>3%</td>
</tr>
<tr>
<td>Learning curve</td>
<td>3%</td>
</tr>
</tbody>
</table>

To discover the actual costs related to training, we obtained historical data, provided by the person who manages internal/external training and courses for employees in the ICT department. We used historical data from January 2006 to May 2012. We were able to select those courses which were related to training in the use of UML, and separate them from other related topics (like Object Orientation, RUP, etc.). The total amount of money spent by the company on UML adds up to 24,313€ in a period of six and a half years (which is approximately 3,750€ per year). A fair number of interviewees mentioned that they did not need a course in UML, because they had had training in the use of UML during their tertiary education. This amount of money invested on UML training is small, compared to the total budget of the department.

On the other hand, a good number of interviewees did consider that training in the use of a modelling tool should be a potential cost factor. They also thought that the lack of training in the use of tools would mean a loss of the possible paybacks that the use of the tool might bring. In the ICT department, 80 people had been trained in the use of a particular UML tool. Buying/licensing a tool is reported as a cost factor by
32% of the interviewees. Based on data from the company, we can calculate that tool-licensing costs around 18,000 Euros/year.

Another investment which was often mentioned by interviewees is the cost of the ‘migration’ of the current situation (especially of documentation) to the new one; i.e. updating the existing documentation to include UML diagrams where there are none, or where another notation is used. Formally speaking, this is related more to the introduction of UML than to the use of UML, yet it is potentially a major investment for companies that are maintaining a large set of systems. Most comments related to migration came from people who are currently working on non-UML projects, and who would like to introduce UML, but who consider the migration of the documentation to be an impassable hurdle. Figure 6.9 summarises these findings and relates them to our theory.

In addition, the interviewees mentioned other costs related to the presence/absence of UML diagrams; these costs are less tangible. Furthermore, the interviewees stated that projects incur costs also because of poor documentation and modelling.
• Miscommunication: although there is a common knowledge of a standard modelling notation (based on training at Universities, or specific training at the company), there are also costs related to miscommunication: “During my study I had a course on UML, but I think it was a bit outdated, although the basics are still the same. Then I started work in this department which implements UML too, in part, so you do get to know different diagrams and how to use them. But even though I think most people have received the same training, there is still a lot of miscommunication, often. So even having a formalised diagram does not necessarily mean that everybody is on the same page. If you look at them all.” [Int14].

• Lack of modelling: When there is no up-front design available from the beginning of a project, there are greater probabilities that a refactoring later will be needed later in the project, because the requirements were not completely understood. The cost/penalty for repairing/refactoring the implementation is more expensive than doing lean up-front design: “I’m convinced that if you did not use UML in the course of a big project, you would have to do a lot more refactoring; so in UML diagrams or in graphical diagrams, the structure of what you are going to do becomes quite clear and you can easily see if there are things missing. It’s a little bit like you have a blueprint and you are going to build a building; you need a blueprint, to tell you what to put into the foundations, and then what to establish what you have to do after that, and after that, and after that. It determines more or less the order of your development too. If you don’t have a plan of your building, and someone starts to build, it will at some point collapse for sure.” [Int26].

• Moreover, when no up-front design is available, the probabilities of defects finding their way in are higher. If a defect that leads to operational defects is left in the software, then this triggers a chain of activities that also involves time from the helpdesk. That may also have an impact on business processes: “... there is no upfront design [...] we directly jump to the code, we test everything. That’s how we started. We failed miserably.” [Int24].

• Lack of up-to-date documentation: One of the comments frequently made in the interviews was that people (especially developers and testers) needed to call other people in the organisation to discover whether the documentation was up-to-date, because they were not sure if that was the case [mentioned by Int36]. This leads to costs in effort, but also to significant delays in time if the person cannot be reached immediately. This lack of one of the most critical quality attributes (up-to-date-ness) is aligned with the results of previous studies (Garousi et al., 2013). In relation to this, we asked some of the individuals interviewed about their preferences for software documentation. We posed them a hypothetical dilemma: They would have to choose between two different projects: the first one is a project which contains UML diagrams in the documentation, but the documentation might not be updated; or the second one, which is a project with updated documentation but without UML diagrams. Two thirds of them chose the UML project, while the rest (one third) selected the updated project. Those who chose the updated project
argued that UML diagrams could be generated easily. Those who preferred the UML projects argued that they prefer to have a quick overview of the system and then go to the code for details about the current situation.

A summary of the perceived cost factors of modelling is summarised in Table 6.3.

<table>
<thead>
<tr>
<th>Costs of Modelling</th>
<th>Magnitude of cost</th>
<th>Costs of not modelling</th>
<th>Magnitude of cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training</td>
<td>Low</td>
<td>Possible misunderstanding</td>
<td>Medium</td>
</tr>
<tr>
<td>Tooling</td>
<td>Low</td>
<td>Spending time reading code</td>
<td>Low</td>
</tr>
<tr>
<td>Creation and update</td>
<td>Medium</td>
<td>Possibility of defects being incorporated</td>
<td>High</td>
</tr>
<tr>
<td>Migration</td>
<td>High</td>
<td>Need for refactoring</td>
<td>High</td>
</tr>
<tr>
<td>Change people’s mind</td>
<td>Low</td>
<td>Cost of reverse engineering</td>
<td>High/medium</td>
</tr>
<tr>
<td>Change process</td>
<td>Low</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### 6.4.3. Process

Here we present the findings related to the element “PROCESS”. We provide the interviewees’ opinions on the relationship between their development processes or SE methodologies and the presence of (UML) modelling. This subsection contributes to answering RQ1. After doing that, we delve into how documentation is used as part of the process of maintenance of a system. Firstly, some general hurdles in relation to documentation are highlighted (this contributes to answering RQ3). We then summarise how the documentation of a maintenance project is used, looking at its usability/usefulness and how it is maintained. It should be borne in mind that we are talking about documentation in general in these subsections, but UML can be considered part of that documentation, so the findings can be extended to apply to UML also, and in fact some of them are specific to it.

#### 6.4.3.1. Relation between development process and modelling

During the interviews, we found that the approaches that are principally used at the company are the waterfall and agile approaches, like SCRUM (75% of the projects are waterfall, 25% agile). In this subsection, we share insights into the maintainers’ perceptions when we asked them about the relation between development processes and modelling:

In waterfall projects, there is a lack of communication or discussion. The information flows in only one direction, and some information is lost on each step. Waterfall approaches therefore lead to less effective solutions: “If you do pure waterfall style, somebody gets a design approved and then gives it to some other guy, and the other guy says: ‘Well I looked at your design, but it's not going to work.’ And then the first guy says ’Hey, it was already done and it has been approved, so you have to build it this way.’ Yes, but it will not work; so there's no discussion there,
which is an even more ineffective approach than if you were to talk about it first.” [Int16]. The perceived quality of agile projects therefore seems to improve in comparison to waterfall projects, on the basis that: 1) everybody agrees on the solution and 2) there are multidisciplinary points of view when constructing the solution. It is thus possible to state that people are more open to revising the design in agile projects.

One interviewee mentioned that the use of UML is not compatible with agile projects, and he would not recommend using it in that kind of projects [Int6]. Nevertheless, his colleagues did not agree with him, because they considered that modelling is also a good practice in agile development. Modelling can also be carried out in an incremental manner [Int24], following the philosophy of agile projects, meaning that using UML does not imply a big design upfront.

We also detected that there is sometimes an incorrect usage of the development methodologies when there is an effort to save time. This leads to a mix of waterfall and agile projects: “In waterfall we have the big design upfront, so first you do a full initial analysis, and then design: when everything is clear you start to develop. That’s the most formal approach in waterfall, but nobody does that. So when a project starts, the waterfall does document, but the early stage of development then starts on the basis of the things that are clear at that moment, so we try to mix a little bit of waterfall and agile. But it’s not the ideal situation” [Int26].

It was also detected that agile projects suffer from a lack of documentation. The main documentation is carried out directly in the source code as comments. This is aligned with previous research (Garousi et al., 2015), in which maintainers have stated that they prefer to refer directly to the source code itself and rely on source-code to support their information needs for maintenance tasks. This way of working implies difficulties for newcomers to the project. There is also a clear need to name conventions and use self-descriptive names in the source code. These problems are magnified when the code gets bigger and more complex. The interviewees considered that the lack of documentation is complemented by a face-to-face communication and by obtaining more experience on the same project [Int33]. But in the agile projects, maintainers also argued that the updating of the documentation should be part of your ‘definition of done’ [Int24], otherwise it would never be done.

These findings about the influence of the SE process in the modelling approach are summarised in Figure 6.10.

Finally, on this point, we discuss some remarks regarding the relationship between modelling and the different phases of a common software life cycle. Modelling is usually done during the early stages of software development. There is a clear relationship between the production of source code (during development or during maintenance) using all the available documentation, including the presence of UML diagrams, because UML diagrams are used to guide the development of the software. The interviewees also stated that there is a clear relation between modelling and the software testing phase, because UML diagrams might be used to guide the testing use cases. They considered that one of the main issues of testing is to check the alignment
of the source code with the diagrams. If the source code reflects what was represented in the diagrams, the quality of the system is ensured.

We also found problems in the investment made during the different parts of the software life cycle. Very often, the investment made in the development phase is insufficient. The reasons for this could be:

- There is a lack of time to finish a project punctually, which means that the construction starts before the end of the definition of the requirements or before the completion of the modelling of the system.
- Sometimes companies that work as suppliers make a bid on a project and allow low development costs in order to ‘win’ a project when competing with other companies. After doing so, this supplier company builds up unique expertise in a particular system and is then in a good position to also be awarded many subsequent maintenance projects. The supplier company therefore acquires a stream of maintenance projects.

These two practices are carried out following “unwritten” rules of the company, although the interviewees mentioned that solving a problem in the design phase is cheaper than solving it later, during the maintenance.

Furthermore, there are projects which invest all their resources in the construction of the system while the company simultaneously attempts to invest a minimum amount in the maintenance of these systems. As a result, such systems end up being extended in a quick-and-dirty manner, by means of “patches”.

All of the above leads to longer maintenance projects that are sometimes difficult to change owing to the ad-hoc means used to construct the system. We would therefore like to state that cheap developments sometimes lead to unmaintainable systems.
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

6.4.3.2. Documentation

Although our interviews focused mainly on UML, some interesting conversations about software documentation in general also arose. This was due to two factors: 1) we treated UML diagrams as part of the documentation of the system, and 2) those who did not use UML talked to us about documentation in general. The responses to those interviews allowed us to extract several hurdles that hinder the effective use of documentation in general. The majority of these hurdles (80%) show that the use of modelling is not only a problem of modelling itself (per se), but also of the overarching technologies used to handle and navigate documentation:

- There is a high level of duplication of documentation in different storage systems: Alfresco, Jira, Confluence, Dropbox, Shared folders, network driver, etc. This makes the documentation difficult to manage, to search in, and to update. A reduction of storage facilities and standardisation would be needed. Otherwise finding documentation for a system is frequently a time-consuming activity.
- Different projects use different structures (directories, naming) for stored models (and often files in general). This again complicates the accessibility of information. Standardisation would be needed to solve this problem.
- People experience difficulties in searching for a specific document/information. This might be caused by the two previous points on this list. The problem related to the fact that the information required is believed to be contained in available documentation, but cannot be found has already been highlighted in previous research (Lutters and Sean, 2007). Information that is needed is often ‘buried’ in, and frequently scattered throughout, voluminous documentation.
- For ongoing projects, their documentation may not yet be available in the shared repositories, while peer projects might already need to consult it.
- Techniques used for cross-referencing between documents are not reliable (there are sometimes shortcuts to non-existent paths, so they are ‘dead links’).
- Once the documentation is found, the reader needs to verify that the documentation is up-to-date, and consumers therefore need to spend time (especially making telephone calls) verifying whether documentation is up-to-date.

Each of these issues by themselves constitutes a critical hurdle as regards ensuring that documentation is kept up to date - critical in the sense that if the documentation cannot easily be found, the engineer will not bother to use it and will therefore not update it: “So of course it’s not always difficult to get the data, but then you really need to know the people. That means that you are depending on people to get information, instead of depending on a good system, a good document system where you get the data yourself. That’s the thing.” [Int32].

Storing all the documentation in a single document would not be usable, because dividing documentation is necessary for reasons of size and abstraction. However, this division is hindering the practical use. The partitioning of information across multiple documents makes it more difficult to keep information consistent and up to date: “I hate jumping between documents. That makes it hard to have an overview of what’s
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

what” [Int12]. Another reason for dividing the documentation is that of presenting different views to different stakeholders: “The core development team consists of 3 developers, one information analyst and one project leader; hence 4 ‘technical’ people. In addition to that, there are eight different stakeholders involved in the project (deployers, databased admin, maintenance, business analyst, or project architect). Some of them are informed; others need to give different types of approval. Now for whom should the documentation be optimized?” [Int13]. This practice was also detected in previous empirical studies (Petre, 2013).

Even if a document can be found easily, there is another problem: there is strong agreement amongst the interviewees that it is hard to find the relevant information within a document:

- Sometimes it is difficult to find information within documents. This problem should be solved if we are to facilitate the effective use of documentation.
- For example, business rules are not put in UML, but they do influence the design [Int26].
- Often the ‘why’ (rationale) is missing [Int23, Int27].
- It is not easy to search in diagrams/models [Int37, 28].

Another problem detected, also related to the documentation is the fact that it is based on projects rather than around systems: “You would like to have a distinction between system documentation and project documentation, and both have a valid position throughout the documentation, but that is not really the case at the moment, unfortunately” [Int1]. This increases the traceability of changes per release, but it complicates the understanding of the system as a whole. “Well, it depends; for example, project documentation [...] nobody really reads it, as there are only a few documents that are really interesting, so...on one hand there’s too much documentation [...] So for the....to keep documentation, [...] if a process is changed in the business or if a software is changed, not always has the documentation also been changed, so I think there should be a way where you say <<Ok, this is part of it as well, so change the documentation too, including a review, so that it’s better managed>> not only manage the system but also the documentation. There I see a lack sometimes, but the project documentation is sometimes just too much. ..... That usually takes too long; it’s too much, and nobody reads it.” [Int32]. What is more, processes do not enforce the updating of documentation when software is changed. This is supposedly compulsory for all changes, but it is completely unmanaged as things stand. Having some type of process control could be good (and would be easy to implement in a tool).

As a reflection which could be extracted from the problems mentioned in this section at this point, we would like to point out that the use of modelling is not only a problem of modelling itself (per se), but also of the overarching technologies used to handle and navigate documentation.

In this section, we have already discussed several issues related to the documentation of a project. Nevertheless, there are other issues containing sufficient
entities to be treated independently. In the following subsections we would therefore like to highlight findings related to the maintenance of documentation; these are findings about how the documentation is being used as part of a project, as well as about the usability/usefulness of the documentation itself.

6.4.3.2.1. Maintenance of documentation

We also asked the interviewees about the maintenance of the documentation (in general, not only the diagrams). It is very surprising that the maintenance of documentation took place in only 39% of the projects on which the interviewees had been involved, although this result is aligned with that obtained in a previous survey (Forward and Lethbridge, 2002). The interviewees stated that some of the reasons for this might be the following:

- It is not clear who is responsible for maintaining the documentation.
- There is a lack of compliance to processes. In addition, there are sometimes organisational or process problems blocking the maintenance of the documentation.
- There is a feeling akin to “don’t care’ about maintenance”. This is an attitude issue, and maintainers considered the documentation to be a low priority task. When updating documentation is not a task that is enforced, it will be relegated. This has to do with the fact that the benefit of documentation is felt by people other than those who create the documentation; the creators of the documentation therefore consider it a burden.

- Small changes are not represented in the documentation (especially in the diagrams). Small changes could be maintained in the text of a document, but it is very unusual to have the need to update a diagram. Where a diagram might be changed for a small maintenance task, it would be the sequence diagram. The same occurs when a technical modification is made. That means that only structural changes (which are not frequent) are reflected in the documentation.

- There is a lack of time, which does not allow team members to “spend time on secondary tasks” (time pressure). Sometimes they need to make “urgent” changes, and these are not reflected in the documentation. If no time can be found to update the documentation while the maintenance of a system is being carried out, it would be useful to update the documentation after the project has finished. However, there are no incentives to report out-of-date documentation, or any processes with which to do so.

- Team members tend to memorise the systems and leave the documentation. They do not need to update the documentation, because all the information is “in their heads”. This then leads to a problem of knowledge evaporation when a team member leaves the project: “I think lots of information is also in the heads of the people working there, which was also the case in our mainframe department, but there was always a focus that it should not be only in the heads of people, because if people go away ...” (a perspective from an information analyst and developer who moved from waterfall to agile) [Int35].
Some interviewees argued that the stakeholders do not want updated documentation. They only pay for a maintained system, and they do not care about the documentation. In such a setting, a team finds itself applying all its resources to the source code, rather than investing a small part in maintaining the documentation.

Several reasons were given for not maintaining the documentation, but the maintainers also know that not doing so has several risks:

- Documentation which is not updated becomes an incorrect basis for the maintenance in the future “[...] but in the end it becomes a problem because no one knows how the system works, and so some developer needs to look into the doc and from the doc, to determine how it works. So then at some point, the diagrams and the code work totally differently; then you have a problem with everything, with your maintenance, because existing diagrams are also determined to be a basis for making a code for further enhancements of a system” [Int26].
- Non-updated documentation might lead to mistakes becoming incorporated into the source code “For example, let’s say one year the team does diagrams and the code religiously; they give them ‘in sync’. The next year, for some reason, nothing is in sync, due to project pressure, and the 3rd year, if someone wants to refer to the diagram and then goes on with coding, there will be issues. So yes, it is helpful if kept in sync with the code” [Int37].
- Another important point is that when the documentation is not properly updated, problems related to low traceability between software artefacts could arise “I’m reading this but it does not correspond to what I’m seeing on the screen. And the only thing we have left is to ask a developer how this piece of functionality is implemented, because I cannot find it anywhere. After this we can make a new specification, but it really also delays both the maintenance track and the project” [Int14]. This finding agrees with the survey of (Nugroho and Chaudron, 2008), in which respondents highlighted the importance of maintaining the correspondence model-code. They argued that, from a software maintenance perspective, we lose the main benefits of models as an important source of architectural information when software models no longer correspond to the implementation code.

The reasons mentioned for not updating the documentation (including or not UML diagrams), and the risks of skipping this step are summarised in Figure 6.11.

We conjecture that the difficulty involved in updating the documentation has an inversely proportional relation to the level of detail (LoD) of the documentation. When documentation (or a diagram) is represented using a high level of detail, the understanding of the system is increased, but the ease of updating it is decreased, and vice versa. Projects need to find their ‘ideal’ level of detail which might balance those concepts. Different levels of abstraction in documentation are found because of the fact that different versions of the documentation are created for different purposes. Low level of detail is created to satisfy the mandatory process and also to communicate with the business stakeholders, while documentation with a high level of
detail is created for (more technical) communication within the maintenance team. This shows us that different people need different LoD.

Figure 6.11. Summary of reasons for not maintaining documentation, and risks involved in not maintaining documentation.

6.4.3.2.2. Use of documentation

We also investigated how the documentation is being used. We detected first of all that the value of documentation decreases for some stakeholders when their expertise with the system increases. When a maintenance person needs to understand a system that is new to him/her, s/he first starts to read the documentation. In most cases, source code is a secondary source by which to understand the system (except in the case of a couple of developers, who prefer working directly with source code). This contrasts with the results obtained in previous research (Garousi et al., 2015), which show that maintainers prefer to refer directly to the source code itself and rely on source-code to support their information needs during maintenance tasks. We might, therefore, think that documentation is only used to ‘get familiar’ with the system: “Once a maintenance engineer is familiar with a system, he hardly needs to look up information in the documentation.” [Int18]. When a developer is not familiar with the system, the documentation is consequently perceived as useful. But once the same developer has become experienced with the system, s/he has a mental model of the system in his/her mind and uses it as a basis on which to work. Once in the latter state, documentation is perceived as less useful. These results are similar to those presented by (Garousi et al., 2013), who mention that during maintenance, documentation is used on average in only 18% of cases, while code is used with a frequency of 50%.

When focusing on the use of UML diagrams as part of the software documentation, we detected that maintainers sometimes create their own diagrams. They do this for their own use to describe the system they are maintaining and to summarise what they see in the source code. The purpose of these diagrams is not, therefore, to document...
the system, and as such this type of diagrams is not stored: “I describe a design that I plan to implement [...] and sometimes I use an enterprise architect to draw some diagrams, UML diagrams, but we don’t really document a lot for development.” [Int19]. This is in the same direction as the results of Garousi et al. (Garousi et al., 2013), who highlight that design documents do not provide that much benefit during maintenance. In this case, we detected that the design is not documented and that this would not be a problem for the maintenance tasks. However, the UML models are documented and used during this phase.

Architects can use the UML model to check that the implementation conforms to their intention. “UML is your reference of what it should be, but you have to check if the code that is delivered is in fact aligned with your UML diagram. [...] At some point in the project, we did not do that functional test, but I also reviewed the code, to see if the structures were as I intended them to be according to the diagrams.” [Int4].

6.4.3.2.3. Usability / usefulness of documentation

We observed that the size of documentation matters. Moreover, the size of documentation has an impact on its usability. A large number of pages is considered impractical [Int18]. Some people indicate that 20 pages is already considered to be a lot. Another important factor that influences the usability of the documentation is the language in which it is written. Sometimes companies tend to force their employees to use English as a common language in the company, yet employees would prefer to document in their native language: “I think documentation or source code of programs should be in the language that we speak and that we can read and that we feel, and this is not English. I call it “Denglish” (Dutch-English). What another Dutch person means when he stated something in English is sometimes very confusing; then you read it and you think ‘Oh, well, I thought you meant that, and not this.’” [Int36]. That freedom to use the local language would not be possible in the case of offshored projects.

It is also noteworthy that the usability of documentation is perceived differently by different people and throughout different phases of the system’s development [Int26]:

- Architects prefer high-level overview pictures.
- Integration architects believe that sequence diagrams are the most important type of diagram [Int26].
- Programmers spend most of their time looking at source code. But when programmers look at diagrams, these need to include sufficient detail to allow programmers to relate the diagrams to the source code they are working with (i.e. high traceability is required).

We also detected that UML diagrams would be more helpful if they were executable models. This beats UML when that is used as box-and-lines: “There was one project which used a BPMN[1] engine. With that you can model the business

[1]BPMN stands for Business Process Model and Notation
process in the tool and actually execute it. And it was also very helpful to have another way to know how it should work, as a reference. That beats any UML diagram, because UML doesn’t execute, it doesn’t work, it is not physical; it is just box and lines.” [Int4].

When focusing on the usefulness of the documentation, we detected that if the system (or part of the system) is simple, then no documentation is created [Int18]. Maintainers mentioned that they only document important things. We asked them next about their definition of “important things”. Maintainers agreed that a part of the system should be documented if: it is a complex part, or if it is a critical part, or if the part is not obvious [Int16]. Other reasons for deciding when to document something or not are agreed by the team. Moreover, if the system’s life is expected to be short, not too much information is documented.

### 6.4.4. Practice or style

These subsections contribute to answering RQ1. A summary of the findings obtained as regards practices or style is presented in Figure 6.13.

#### 6.4.4.1. Diagramming practices: (standardized) UML or freeform

We asked our interviewees whether they used UML or other graphical notations during software maintenance. Some of the interviewees mentioned directly that they use UML, and others did not (Table 6.4). There were those who said that they know the UML diagrams are being used by their colleagues, and they know this because, for example, some of them have printed copies on the wall. Some others considered screenshots as graphical notations to communicate layouts on interfaces.

<table>
<thead>
<tr>
<th>Presence of diagrams in documentation</th>
<th>% of interviewees</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains diagrams</td>
<td>77%</td>
</tr>
<tr>
<td>UML</td>
<td>94%</td>
</tr>
<tr>
<td>Non UML</td>
<td>6%</td>
</tr>
<tr>
<td>Does not contain diagrams</td>
<td>23%</td>
</tr>
</tbody>
</table>

Almost a quarter of the people interviewed (23%) considered that the documentation they use does not contain diagrams at all, as opposed to 77% of them who considered that it does. Of those who use documentation containing diagrams, 94% of them stated that they use UML diagrams, as opposed to 6% who consider that they use other notations.

The most commonly-mentioned UML diagrams are: sequence diagrams (69%), class diagrams (55%), activity diagrams (36%), and deployment diagrams (23%). Other UML diagrams were also mentioned, but in lower proportions: i.e., collaboration diagrams, component diagrams, package diagrams, statechart diagrams and use case diagrams. These results are in line with previous surveys (Dobing and Parsons, 2006; Fernández-Sáez et al., 2015b; Hutchinson et al., 2014).
It is surprising that some interviewees referred to diagrams that are not part of the UML set of diagrams when asked about the UML set. Those most frequently mentioned are data flows and data models. The nuance aspects of notations are not used, and developers largely use the common concepts: “I have seen so many names for the same thing. [...] UML or ERD; it is all the same, in my opinion” [Int13]. There was also a common confusion when talking about use case diagrams, because some users considered them to be a table summarising a use case like a “diagram”. This is based on the general practice of describing use cases by using a table.

We also asked UML users about the LoD of their use of UML diagrams. There were more respondents using high LoD UML diagrams rather than low LoD. Some developers also mentioned that the decision about how many details should be in a UML diagram is an architect’s decision, and developers are not taken into account. This finding may indicate that there was uncertainty amongst developers as to what extent that freedom could be exercised, because a model should explain how the system works without allowing programmers too much freedom to determine implementation details, as highlighted in (Nugroho and Chaudron, 2008). On the other hand, there are developers who have access to UML diagrams (either from an architecture/design diagram or from a previous stage of development), but they do not use them because they do not have sufficient details. Developers are used to working with the source code, which contains many details, so they draw their own diagrams based on the status of the current source code in order to have a more detailed diagram, compared with the one created by the architect.

In summary: the use of graphical diagrams is very common. Within this, the use of UML is common. Moreover, the UML notation is used to represent diagrams from other design paradigms (ER, data-flow and context diagrams). Different stakeholders have a variety of purposes with models, and as a result use a range of levels of details in their models. This practice (adapting the diagram depending on the audience) is common in industry, as presented in related work (Petre, 2013).

6.4.4.2. Influence of UML usage on quality of software

We asked the interviewees about the quality of the final product and its relationship with the use of UML diagrams: “Do you think using UML has an impact on the quality of the final product? How?”

In this case, the respondents considered the quality of source code related to performing correct testing and obtaining positive results from it; i.e. obtaining a source code that is aligned with requirements and design: “Quality is the result of checking the result too, so UML is your reference of what this should be, but you have to check if the code that is delivered is in fact aligned with your UML diagram” [Int4]. This is in harmony with the results of the survey presented by (Nugroho and Chaudron, 2008).

Employees working on projects which do not use UML diagrams commonly believe that the presence/absence of diagrams is related to a high/low quality of documentation, respectively. It is very important to note that there is almost a general
consensus amongst all interviewees that the use of UML improves software quality (89% agreed, and 11% believed UML is not related to software quality). The reasons given in support of a relationship between use of UML and high quality are:

- The use of (automated) modelling tools improves productivity.
- The sharing of knowledge is improved.
- A peer-review process makes it easier (and highly recommendable) to adopt when UML is available.

In relation to software quality, we also asked the interviewees about the possible relationship between the use of UML diagrams and the presence of defects in the code of the system:

A couple of interviewees considered that UML usage reduces the phenomenon of defects managing to get into the code of the system, i.e. UML prevents defects, while another person replied that they believed that UML increases defects (but only whenever they are not updated). This contrast indicates that neither the positive nor the negative effects are perceived to be very significant. It can also be seen that some interviewees thought that there is no relationship between software defects and UML in itself; the defects are caused by an incorrect solution, but UML is not the problem.

More than half of the interviewees were of the opinion that the use of UML is helpful when searching for the cause of a problem in the source code. Sequence diagrams are especially valuable for this purpose: “Sequence diagrams are also very useful, for example, if there are bugs or issues on-line and we don’t know how the bug works, how the bug exists there, which components are affected and need to be looked into.” [Int11]. This is in contrast to the results of a previous survey (Nugroho and Chaudron, 2008), which shows the respondents’ indecision on the impact on quality of using the UML on software testability and correctness (defect-count). Respondents in that survey might not have been exposed to testing of plans or criteria constructions using UML models, in contrast to the interviewees in this study (one of the main purposes of using UML in this study was to test guides). Early and more thorough thinking about the design leads to higher quality of design. Moreover, as stated in a quote above, while this may incur some more effort in the design stage, the respondents were convinced that there is an overall benefit in productivity. We would like to stress that although the quality of the design might be improved with the techniques mentioned in this section that does not have to be reflected in the quality of the source code in terms of an improvement. This is because the quality of source code deals with issues different to those from design, such as following naming conventions, having a correct commenting system, using a correct indentation, optimising the nesting depth of loops, etc.

6.4.4.3. Standardisation

We asked the individuals we interviewed about standardisation in their ways of working. In this case, we focussed on those standards used to document the system and the activity of diagramming. Only 1 interviewee considered that there is excessive
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

standardisation, while 31% believed that there is a lack of standardisation. These respondents felt a need for more standardisation in relation to the following:

- **Naming**: naming conventions for classes, attributes, etc., in code and diagrams.
- **Layering**: it is not clear what the recommended layering of the system is.
- **Style**: There are many issues related to the style of diagramming (and subsequently of coding) which are not clear.
- **Level of detail**: it is not clear at what level of detail systems should be modelled.

“We use different terminology, different naming, different layering” [Int1]. They argued that the standardisation should be established in the early phase of a software life cycle, in order to obtain the maximum benefits during the rest of the project [Int14]. They also emphasise that the standardisation should be done right across all the teams if it is to be successful [Int4]. The interviewees considered that standardisation plays a very important role when third parties are involved [Int12].

Although the needs mentioned above arise when asking for diagramming, some respondents highlighted the need to also standardise the source code (naming conventions are needed) [Int17], and text (what should be written and the structure of the document) [Int25]. The main benefit of introducing standardisation in any of the items of the software documentation that was mentioned is always a reduction of the risk of misunderstanding [Int23]: “A standard is something that will be understood by all people, across all platforms.” [Int28], or “The problem in ICT is that every platform uses its own language, its own way of looking.” [Int28].

Independently of their opinion on the presence of standards at the company, many of those surveyed agreed that there is a lack of compliance with the standards. They justify this, using the following reasons:

- The way they work is intuitive, and they do not need to follow other standards [Int10].
- They have previous knowledge, and they do not need to follow all the standardised processes [Int10].
- They believe that standards do not help in every case, and sometimes these standards are not applicable [Int16].
- They also believe that there are items, such as text which, unlike diagrams, cannot be standardised [Int28].

Those who mentioned that they do not follow standards explained how they work. The majority of them use solely box-and-line diagrams (19%), while others use standard UML, but in a non-strict manner: “Because there is no real standard, people use their own inventions” [Int18]. This finding is aligned with “selective use” of UML detected in previous studies (Petre, 2013), where UML is used in design in a personal, selective, and informal way, for as long as it is considered useful, after which it is discarded. Advanced features of the notation are not used, and do not therefore need to be updated. Moreover, some interviewees use diagrams that are similar to UML, but may not use all its elements correctly (for example the different types of arrows in a
class diagram) [Int3]. In some cases they use the formal UML specifications, but they mix them with their own notation to complement them [Int5], and they complement the diagrams with personalised legends in order to clarify the non-UML part of the diagram [Int23].

The individuals who gave their opinions also noticed a problem with standardisation: when a system is very old (especially in the case of legacy systems), the documentation contains a mix of different standards that had been adopted for several years. We are therefore of the opinion that standards have a short life-time: “I've heard a lot of standards over the last 13 years, and what is standard now isn't so standard in few years’ time, because then there is another standard. [...] What is standard now is not so standard 5 years from now.” [Int27]. One benefit of using only ‘box-and-lines’ in diagrams is that it makes them independent of the evolution of the UML notation.

It was also surprising that maintainers know that the diagram they are creating is not correctly written, but they deliberately write it in that way in order to trigger discussion: “It is better to be unclear than to be misunderstood.” [Int12]. The idea behind this is that a reader of documentation will recognise that a part is unclear, and this triggers him or her to ask for clarification. Whereas a misunderstanding will lead to a wrong (design) decision.

In many cases (63%), Visio is used to create diagrams. Visio supports the UML notation, but does not enforce the syntax as strictly as a dedicated UML-CASE tool does. Arbitrary graphical shapes can be connected without regard to their meaning and hence without keeping syntactical rules. Moreover, Visio does not create an actual model of the system, only a diagram. Visio thus does not support consistency inside diagrams/models or across diagrams/models.

It was also surprising that in some cases, the interviewees are following a standard notation but they do not know that they are. They merely copy the way of modelling that is already being used in the documentation: “if sometimes I have to make new diagrams or I have to change them, what I do is just use what is used in that system” [Int27]. This means that the formalised documentation is sometimes produced using previous documents as a basis.

Another problem with standardisation is that it is sometimes very strict, and slows the maintenance process down. There is usually a formal procedure for the approval of milestone-documents. This process typically takes a long time, and thus slows down the development. This discourages people from asking for approval or even from making (small) updates to milestone-documents [Int18].

Mechanisms to incentivise the correct use of standards should thus be introduced: “If you let people choose, you lose all your advantages. So, yes, force them” [Int8].

Sometimes standardisation is not provided by the company, and producers of documentation agree with the consumers of the documentation on the meaning of their diagrams (it would appear that this occurs in a just-in-time-manner). Once common
agreement is in place, this is an effective way in which to work: “Because there is no real standard, people use their own inventions” [Int18]. The only risk to this is a change of staff. This practice of using a common agreement about meaning takes place because the team members, and especially the business stakeholders, sometimes lack knowledge [Int23].

We found that there are also some concerns regarding the use of documentation as a standard step of the maintenance process that are related to the team members’ attitude.

• Some developers do not like documenting, and they consider that this is the responsibility of other team members: “I am not a writing person, I am a building person. I hate documentation because I am a technical guy. [...] We are not documentalists. We are builders. That is a different state of mind.” [Int13], or “Developers don’t like to write documentation” [Int6].

• The documenting part of the project is considered to be a boring part, and maintainers try to avoid it: “documentation is almost a dirty word you should not use.” [Int35].

• From a selfish point of view, maintainers believe that documenting is work done only for others “If I write documentation then I am helping a colleague.” [Int13].

Previous research (Lutters and Seaman, 2007) has highlighted that documentation that is written from the perspective of a maintainer (and is sometimes even written by a maintainer) is especially useful. It would, therefore, appear to be important to take this point of view into account.

• Sometimes developers do not follow the specification provided; they do the maintenance and then implement what they consider to be a better solution: “often they [developers] just don’t listen. They just do it the way they think they should do it.” [Int1].

A summary of the findings about “PRACTICE” discussed up until this point is given in Figure 6.12.

6.4.4.4. Use of reverse engineering

During the interviews, we asked the subjects about the use of reverse engineering, especially when diagrams are not available. The first observation is that sometimes people do not know that a (UML-case) tool can help them to automatically partially extract a diagram which can be used as a basis for creating UML diagrams.

In the case of projects for which UML diagrams are available, the diagrams (along with other documentation) are in many cases not updated according to the changes in the source code. In these cases, the interviewees considered the use of a reverse engineering technique to be a very helpful tool in obtaining a trustworthy diagram. We found that reverse engineering sometimes takes place, but this process often requires additional manual work. Indeed, the interviewees considered reverse engineering to be a lot of work: “That’s a nasty question, because the documentation should preferably be up to date. You want the documentation to be correct, so try to get documentation
up and running up to the situation as it is. […] In practice this is a very tedious job, because you have to be really specific if you want to describe a piece of software. It basically means digging into the code and seeing what is going on exactly here, and here. As I’m not a Java expert this really delays the process, because you need both kinds of people to reverse engineer the documentation. That means it is usually not done” [Int14].

Figure 6.12. Summary of findings about Practice.

Alternatively, when no graphical documentation is available, UML diagrams are manually RE to attain an understanding of the source code [Int19]. We therefore found some cases in which people carry out automatic reverse engineering, but after the automatic process, a manual process is needed, when the architect tries to “clean” the diagrams. The people who responded considered this process to be difficult, because they were not able to deduce the implicit policies that were used to generate that specific diagram, or why something is structured in that way, or if a design pattern was used: “You have to go through the code in a rather labour intensive way and look at the classes most of the time. You have to try to deduce, on the one hand, the business concepts and the patterns that have been applied, and the policies that have been driving the design, how clear any policies were, and then you make the diagrams. Of course you can do automated reverse engineering of all classes that have been implemented. But this would also produce the framework classes, so the level of detail is extremely high if you do automatically. It is actually useless to do it like that. You therefore have to go through the code as a human being and try to deduce the intelligence that is in the design, and then make the model. In that sense it is less useful than if you have applied it from the top-down approach” [Int1]. In reverse engineering “it is hard to understand why a certain function was there, or is not there at the present time” [Int24].

Although the “cleaning” process is carried out by the architect in the early phases of the maintenance project, there are developers who would like to have the original
reversed engineering UML diagrams, because these are more traceable to source code. In such cases, they would consider the UML as the “truth”. However, this approach loses followers when the projects are very big and diagrams become unreadable.

We asked the UML users if they would prefer a diagram which originates from a forward design process or from a reverse engineering process. In most of the cases (70%), they prefer the forward design processes.

Some other problems found with Reverse Engineering are listed below:

- Models capture information that cannot be extracted from the code. This is an architect’s perspective: “Reverse engineered models are completely not useful. Too detailed. Not the right semantics. No abstraction.” [Int4].
- Sometimes the RE diagram is drawn up by a single developer; the design is not therefore discussed and hence not disseminated between multiple developers in the team. Reverse engineering eliminates the growth of a shared understanding [Int4].
- Programmer’s perspective: “I prefer the reverse engineered diagram over the diagram from the architect because it is easier to update with the changes that I make to the code. [...] To get an overview I don’t recommend using RE, except when there are no forward diagrams.” [Int11].
- FD and RE design models may use different naming for class names, methods and operations. As a result, it is not easy to use them together/merge them: “reverse engineered diagrams are a bigger truth than the forward engineered diagrams, because ... you may have different names of objects in the design and in the code, but in reverse engineering there is no choice, you just import your code, so there is no way you can have this.” [Int37].
- RE diagrams contain too many details: “there are properties of an object, which I will never model; I just say it has a list of properties. Then in a reverse engineered model, I get like 10 properties actually, and so I have the problem of ‘Oh what is this anyway?’” [Int24]. “The detail of reverse engineered diagrams is not very usable” [Int3]. “For projects with lots of code you get lots of diagrams, with all the details, but they are not really readable. [...] If these tools generated some diagrams with less details, that you could specify how detailed it should be; then maybe they would be useful.” [Int19].
- RE diagrams do not recreate the desired layouts: “I could reverse engineer from an SAD to UML, but then the whole thing is: it has to be rearranged” [Int3]. The layout created in forward designs contains semantically-meaningful information for the designer. For example, related classes are close together (even if there is no association between them).
- Reverse Engineering is time consuming: “it took me ages to do reverse engineering and then figure out those sequence diagrams.” [Int37]. This relates particularly to extracting/abstracting dynamic information.

In summary, we conclude that:
• For programmers, reverse engineering may be a practical way to obtain a diagram that describes a fragment of interest of the system in hand. Yet programmers prefer the design to be updated automatically.

• For architects/designers, reverse engineering is not considered a practical option because:
  o It yields diagrams that are too detailed and that require significant efforts in manual processing. In particular, details need to be left out, and a meaningful layout needs to be created.
  o The Reverse Engineering functions of current UML case tools fail to recreate dynamic/behavioural views of the systems.
  o Using Reverse Engineering to recreate a design model after the facts (i.e. after (much) of the programming has been done) may result in the omission of discussion about the design that would otherwise increase the shared understanding of the design of the systems amongst the developers.

Figure 6.13. Summary of findings about Style.

6.4.4.5. UML versus other graphical notations

We asked the interviewees to compare UML with other notations which are in use at the company and which might be used instead of UML.

Some of them referred to Archimate as an alternative notation to help in system modelling. They were of the opinion that Archimate is a better notation for business models [Int2] than UML, but that UML is more useful for modelling functional documentation. UML is also considered to be a modelling language that is easier to discuss (especially for developers) than Archimate [Int26]. One advantage of using Archimate rather than UML is that Archimate comes with some guidelines for design,
such as layering to help in design [Int1]. The disadvantages cited by the interviewees are that Archimate is less powerful, and too abstract [Int23]. Nonetheless, one person considered that the use of UML in combination with Archimate would be very beneficial [Int1]: “UML is easier, so I would say in discussing, while designing or building systems, or in making the preliminary design steps it is better to have UML, because it’s easier to discuss. But if you are working on the border-line between say, hardware, system software and software, ARCHIMATE is better, I think, because it can warn you of certain problems that might arise if, for instance, you design a heavy application and have only one server. So it depends on your goal.” [Int23].

Other notations were also mentioned as good candidates for complementing UML:

- User-interface perspective/Screen layouts [Int18] and maybe screen-flows.
- Executable specifications (esp. BPMN) [Int1].

Finally, Domain Specific Languages (DSLs) were also referred to. The diagrams generated with this kind of notation are not regarded as adaptable as UML, because they are only able to present one dimension/view of the system. This would have the advantage of providing a better understanding of that view of the system, because it might contain lots of details. But there is the risk that something else might be lacking, because diagrams are usually created from one point of view (structural view, network view, behavioural view, etc.). All the aspects may not be placed together: “But sometimes our projects deliver those messages in DSL itself, which I find so much easier to look at, because you can see it more or less in only one dimension; it’s a very hierarchal thing. In UML you can put things next to each other; you have 2 dimensions.” [Int3].

When we asked the interviewees about the modelling notation, some of them were not able to answer which notation they were using. When digging deeper into this category we discovered that 38% of the interviewees uses arbitrary graphical notations (or box-and-lines), or that only the basic elements of the UML syntactical notation are used in documentation: “Rectangles and lines. It does not have any symbol [...] with arrows, dots or hollow or full circles.” [Int18]. Some of the reasons for following this approach are summarised in the following list:

- Some developers adapt their notation to the one that is already used in the existing documentation. Being consistent with one existing document is of more importance than having consistency in all the documentation for different systems. The documentation of old systems, therefore, typically uses old graphical notations, and continues to do so even after recent extensions.
- Using only ‘box-and-lines’ in diagrams makes diagrams independent from the evolution of the UML notation. If advanced features of the UML notation are not used, then they do not need to be updated either. The fact that the UML notation has had several major and minor revisions of its standard (at least 3 major ones in 10 years), has been a factor in reducing the eagerness to conform to all the details of the syntax.
People that create documentation assume that the reader has a certain level of domain knowledge [Int21], but sometimes this is not entirely true, because the consumer of a document is, on occasions, a business stakeholder.

“You can achieve the same with Visio as with UML, but you need to align how to use it.” [Int26].

We researched the conformance of diagrams to the UML standard(s) in a collection of 35 Global Software Designs at our case study company. This set shows that 40-50% of the diagrams are formal UML diagrams, compared to 60-50% of the diagrams which are not UML. In this last group we found some diagrams of other notations, or diagrams which seem to be UML but which are actually not (box-and-lines, or people’s own inventions).

### 6.4.4.6. Comparing text and diagrams

One important issue that was discussed during the interviews was whether the use of diagrams (especially UML diagrams) is more helpful in understanding the system than documentation that consists only of text. Those interviewees who had been working at the company for a very long time, who had not used diagrams during all their time at the company, and who had not had training on UML diagrams, considered that diagrams would not be helpful for them. We might explain this as being due to fear of change, but also because of unfamiliarity with the notation. In contrast to this group, the majority of those who replied to the questions considered diagrams to be a very helpful tool as regards understanding the systems and the changes which need to be made to them. They based the benefits of diagramming on the possibility of increasing the level of abstraction of representing the system. The following sentence was repeated in the majority of the interviews: “A picture explains more than a thousand words.”

The interviewees who argued in favour of using diagrams because they believe that this improves the understanding of the system, did so for the following reasons:

- People are visually oriented [Int5], and they usually prefer visual notations [Int24].
- Diagrams help when searching for something. It is faster to look for something in a picture than to read a long text, because of navigational issues. This view is supported by the majority of interviewees, although one person did not agree, pointing to the fact that there are lots of tools that support text search [Int37]. A reader can judge quickly whether a diagram contains the information s/he is looking for. Judging this in text requires paying closer attention and thus more time: “People are more likely to skip over a piece of text than over a diagram.” [Int31].
- Text is so much more difficult to maintain than diagrams [Int37]. This may be related to the previous bullet: it is difficult to find the piece of documentation to maintain. Furthermore, it is important to highlight that the maintenance of diagrams requires skilled people (the maintainer, at least, needs to know the notation), but the maintenance of text might well be done by anybody [Int25].
Chapter 6: What is the Impact of Using UML in Software Maintenance in Terms of its Effectiveness when Used in a Software Project?

- Diagrams are easier to understand than text [Int32], although depending on the type of diagram, not everybody might give the same interpretation [Int28]. In relation to the understanding category, we detected that pictures solve problems of understanding for dyslexic people. This is not a very common condition, but it was one that some of the interviewees suffered from [Int13]. Some interviewees also highlighted that diagrams are easier to understand in the context of systems because they describe relations in easier simpler way [Int12]. Moreover, the presence of diagrams in legacy software documentation is very valuable for an easier understanding of the system [Int19].

- Diagrams are easier to compare than text: “It’s easier to put two diagrams next to each other and look at them and see the relation between them. Text?... it’s much harder in that.” [Int12]. This means it is easier to detect what has changed in a diagram than in a text.

- It can also be said that text presents difficulties as regards facilitating the traceability with other documents, or with source code [Int9], because diagrams are better at representing structures: “text just lacks the means of connecting all the components, and it’s also very difficult to keep track of all those dependencies that you have in the environment when somebody is just telling you what there is/are, or is writing down everything simply because you want a quick overview of everything that there is” [Int31].

Some of those surveyed said that text is regarded as cheaper (in terms of cost) than diagrams [Int28], because everybody knows how to write, but not everybody knows how to use a modelling notation. Modelling can enforce systematic use (syntactic correctness), but when using natural language text, much more freedom will be used in writing [Int28]. There is also the issue of information density. Diagrams can convey a lot of information. This is good for abstracting, but may be easily misunderstood by non-experts in the notation [Int12]. Furthermore, creating graphical representations early during development enables such errors to be found and corrected while their repair is still ‘relatively cheap’ (this is in comparison to refactoring such structures later in the project when much more code will need to be checked/changed). The lack of a systematic use of language might lead to more possibilities of errors creeping into text than into diagrams, because creating a diagram entails a more detailed thinking process: “I think you can make a mistake more easily in text than in a diagram, because in a diagram, you have to think more about... While you draw, you can see the mistakes you made in the text, because things don’t match anymore. They can be going in a totally different direction...it’s impossible to draw this when you try to put it down. I think it (a diagram) can help.” [Int10]. In relation to the presence of mistakes in text or diagrams, the interviewees considered that mistakes in diagrams lead to greater errors than mistakes in text [Int27]. If a diagram is not correct, then this is a big error. This may be because information that is covered by diagrams relates to architecturally-significant aspects of the system: “People take [a diagram] as the truth sooner than a piece of text, so they might be less critical towards it” [Int12].
The interviewees also considered it necessary to use text to explain diagrams, as a complement to the graphical information, because diagrams alone are as helpful as text alone: “a picture is mostly saying more than the text alone, but it's the combination, because you have to explain what you want, and that's done in text” [Int10]. Text usually is too detailed in comparison to diagrams and an overview cannot be extracted [Int9]. Diagrams provide a global, high level overview, while text explains rationale and provides details [Int24]. This means that text is considered better for explaining details [Int12]. “Your first thoughts really show where things are changing and different, and highlight the aspect that you need to address. That needs an explanation, because it’s not the final picture.” [Int3]. Text is therefore used to explain/highlight changes, differences from previous designs, and also exceptions in flows [Int34]. We would also like to highlight that the complementary relation of diagrams and text could be extrapolated to the context of oral communication, where talking provides extra information to complement written documentation: “Talking to people can give info about the history.” [Int12], and “When I explain a diagram, I emphasize changes, differences.” [Int3].

6.4.5. Tooling

In the SE community’ discussion regarding the adoption and effective use of modelling, the category of tooling was identified by several researchers (e.g. (Whittle et al., 2013)). In this section, we summarise our main finding that relates to the tool support for modelling and documentation discovered through the interviews. These results contribute to answering RQ3.

Several participants in our case study indicated that they like to mix text with UML diagrams, and also with informal box-and-lines drawings. In reality, virtually all software (design) documents are indeed a mixture of text and diagrams. Unfortunately, current UML CASE tools do not provide any support for this (Chaudron and Jolak, 2015). From a complementary angle, word processors do not support, but rather hinder, the updating of diagrams. We conclude that developers would like tools that allow them to flexibly add text and custom notation to UML diagrams. Yet even today – 20 years after the introduction of UML - such tools do not exist. The need for tools to support the mixing of text and formal diagrams and sketches has been supported by other studies (Dekel and Herbsleb, 2007).

There are several tools that support the management of different versions of textual documentation or source code. But in the case of UML diagrams, the support for versioning is immature, especially in the case of the merging and diffing of models “[when asking about UML’s disadvantages] that we don’t have real versioning on the UML design has more to do with the tool that we use; it’s not related to UML. I think it has more to do to the tool.” [Int26], [Int29]. Sometimes this lack of tooling is substituted by a definition of a human process: “We have the live structure and development structure. So basically what will happen is, if a document needs to be updated you take a copy of the live version, make the changes so that it becomes the development version, put a double tag saying it’s been updated and give the dates it’s
been updated and stuff, and say what’s been updated. And once the codes been done and loaded, it can be moved from there, and then it’s an algorithm.” [Int20]. When working in this way, there is a new task which should be done by a team member, which could be done automatically if there were a tool with which to manage the diagram versioning.

The same lack of tooling applies when the maintainer aims to reuse a model or a part of it. It would appear that UML tools do not support any notion of modularity of the model [Int8]. As a result, it becomes difficult to reuse parts of a model.

There is no specific notation support in UML as regards representing design decisions/design rationale (‘why’s’) or linking these to the actual design. As stated in (Aseniero et al., 2015; Burge et al., 2008; Kruchten et al., 2009), design rationale should be documented to facilitate understanding and maintenance. The explanations of designs currently need to be written in a separate document. There are two key problems when following this approach:

- The high likelihood of losing the traceability between model and text, thus leaving the documentation and model out-of-synch.
- The ‘Dizzying’ of maintainers when they are reading the documentation: “I hate jumping between documents; that makes it hard to have an overview of what’s what.” [Int12].

In relation to this, a common problem of UML tools also arises; there is poor support as regards searching in models.

We also found that if people obtain training in modelling, then this is targeted towards understanding and using the notation, but not towards the use of the tool. Given that the effective use of UML requires the use of advanced CASE tools, it may be wise to also invest in training people how to use such tools, showing their main features but also their details. Some people that regard the tools as intuitive are against this recommendation.

A majority of those surveyed (12 people) state that the use of a tool would help them to correctly model a system, especially because tools help to create syntactically correct diagrams [Int14]. Conversely, one interviewee mentioned that using a tool to model a system is a waste of time; he prefers to create a model on paper/whiteboard and take a picture of it to then attach this to the corresponding document [Int16]. But it is clearly difficult to maintain a model (keep it up to date) in this manner.

The needs of the modelling tools which were highlighted by the interviewees of this case study are summarised in Figure 6.14. These needs complement the list of desirable features of a proper tool for software modelling presented by Forward et al. (Forward and Lethbridge, 2002), who mentioned that it would be useful to have tools with which to track changes in a software system for the purpose of updating and maintaining its supporting documentation. According to the results of Forward et al. (Forward and Lethbridge, 2002), in order to track changes between documents and source code, the technology must be able to relate:
• documents to source code,
• source code to documents,
• documents to other documents (in, for example, hyperlinked environments).

Figure 6.14. Needs of modelling tools detected in the case study.

6.4.6. Context

When able to fit the question into the interviews, we asked about factors that would influence the use of UML in a project. By sampling the respondents, we found that the larger size and complexity of a system are factors that increase the likelihood of UML use.

We also detected some details about the influence on the approach to documentation and modelling of two kinds of maintenance projects which have special requirements. They are the outsourced maintenance projects or offshored maintenance teams. These subsections contribute to answering RQ3.

6.4.6.1. UML and outsourcing/offshoring

Several of our interviewees had worked on maintenance projects which involved outsourcing or offshoring (4 people, to be precise). In those cases, the respondents mentioned a special importance of documentation (especially documentation containing UML diagrams), based on the following reasons:

• To improve communication (mentioned by 3 out of 4 maintainers enrolled on outsourced/offshored projects): “Especially for offshoring teams it is very important that your documentation is correct.” [Int5].
• To guide the implementation: “They are doing the programming based on that notation (UML).” [Int5].
To check the implementation (conformance to design): “**UML is your reference of what it should be, but you have to check if the code that is delivered is in fact aligned with your UML diagram.**” [Int4].

A lack of modelling also influences projects in which multiple parties are involved. This deficiency leads to a corresponding lack of clarity in specifications, which aim to precisely define the work that is demanded from a subcontractor, or to describe an agreement on the scoping of responsibilities between teams.

If we instantiate the element “CONTEXT” of the baseline theory to present the results obtained about those maintenance projects that have an outsourced team or in which the maintenance is offshored, the findings on this subsection are summarised in Figure 6.15.

![Figure 6.15. Influence of outsource/offshored maintenance on the documentation approach.](image)

### 6.4.6.2. Legacy documentation and modelling

We investigated the presence of graphical notations on the documentation of legacy systems. There are a large number of systems that were developed in the pre-UML era. The documentation of these systems does not contain models. The older the system is, the fewer diagrams it contains (also driven by the fact that old computer systems did not support graphical user interfaces). Migrating this documentation to make it compliant with present-day documentation and modelling practices is a huge effort, and hence a huge investment [Int8]. No automated tooling seems to be available for this, either because reverse engineering does not produce the right abstraction, layout and behaviour models, or because there are no reverse engineering tools for old programming languages.

Moreover, for many legacy systems, there is a discussion about when to phase them out (end-of-life) and replace them with new systems. In such cases, companies are even more reluctant to invest in the documentation of these systems, while this documentation could, at the same time, be very valuable as regards building the replacement systems. It should also be said that in some cases the system is expected to be replaced by a new version soon, which means that no more investment is made in that legacy system; sometimes the replacement never occurs, however.

There are occasions during maintenance engineering on legacy systems when developers create diagrams (e.g. for their own understanding or to plan a solution), but
then there are no clear incentives for using these diagrams to update the documentation. Moreover, if diagrams are made, they are often in the same style as found in existing documentation, and hence use box-and-lines or older diagramming notations (dataflow diagrams). What is more, screenshots may find their way into documentation.

Furthermore, some people working on legacy systems are not trained in new notations like UML. So they cannot create UML diagrams. In addition, young people (with knowledge in UML) who are added to legacy projects are constrained to use a notation that all team members can understand.

If we instantiate the element “CONTEXT” of the baseline theory to present the results obtained about those maintenance projects of legacy systems, the findings on this subsection are summarised in Figure 6.16.

![Figure 6.16. Influence of legacy systems on the modelling approach.](image)

### 6.4.7. Other Findings

One interviewee had recently started working in an agile team after spending 15 years following a rigorous waterfall approach in the mainframe area. In that style, documentation had to be complete and signed off before it was handed over to the next phase. He is new to UML and agile development: “... and I say this carefully because I’m new in this department, but I have noticed there is a lot of fixing smaller errors after implementations too, and this is definitely different from what I was used to; when it came to the time to implement [...] anything that was wrong made the alarm bells ring. [...] Now it seems like it’s more or less acceptable to a certain extent to be less strict on this.... or maybe it’s just the way the e-services are connected or how different teams are working, so the chances for such defects or errors is bigger.” [Int35].

Furthermore, during the analysis of the interviews we detected some misperceptions that we consider worth mentioning:

- **Tooling is expensive:** We made an inventory of the tools in use at the company: Visio (15% of people using a modelling tool), Bizz Design Architect (5%) and Sparx Enterprise Architect (80%), taking into account that one person might use more than one tool. The prices of licenses for these tools are between 135€ and 160€; a total of 150 licenses were needed in an ICT department of 800-1000
employees. In addition, between 4,000€ and 6,500€ per year was paid as maintenance costs related to the use of the tools. For this size of an ICT department, the costs for tools are relatively very small compared to the yearly budget (mostly in manpower) of software maintenance projects. Moreover, the costs of tooling are fixed costs which can be paid off fast. Tooling costs were not considered an issue by the management of the ICT department. The perception of tools as an important cost is also highlighted in the survey performed by (Hutchinson et al., 2014). That survey focused on MDE, but the main tool used for modelling at the Company, Enterprise Architect, is a popular tool for MDE.

- An often heard argument in favour of not prioritising the maintenance of documentation is that ‘the business’ stakeholders or clients do not value documentation. While we agree with this, we believe that the use of modelling and updating of documentation is a technique that is internal to the software developers and that can be used to produce software faster and of higher quality. The business stakeholder only sees the shorter development time and higher quality. The same reasoning applies to the following set of arguments: “Business stakeholder does not understand the diagrams. Business stakeholder does not see the value. Business stakeholder does not want to pay for this”. Instead, we would recommend reinforcing the idea that models are for internal use to run the project more efficiently. This contradiction (business view vs. development view) may be linked to a short term vs. long term trade-off: not making documentation at this time may lead to slower and more expensive maintenance on the same system at some future time.
- UML is identified with the RUP- or Waterfall-approach. It is a fact that the RUP is a methodology that is based mainly on UML, and that Waterfall is usually taught using examples of a methodology to introduce UML. But the use of UML is not limited to these two software development methodologies. UML can be successfully integrated into other software development methodologies, like agile methodologies.
- Model-Based Systems Engineering (MBSE) is a technical approach, and hence not a solution for team or organisational units that are not working well together. In general, technological improvements only bear fruit when the team is working well together.
- It was not clear which person/role is responsible for maintaining the documentation. All the interviewees tended to fix their attention on their colleagues to establish which person was responsible for that purpose. The person ultimately responsible was the project manager, but it would appear that he did not state who was responsible for this maintenance task.

Finally, we detected gaps between the interviewees’ opinions that should be highlighted. There are three issues here:

- There is a gap between the architect and the developer. While architects spend a lot of time creating precise and detailed diagrams because they believe that developers
like to use diagrams, some developers prefer to work directly with the source code. But the problem is not based on the notation, because developers also generate their own UML diagrams and introduce more details. The problem of the architect’s diagrams might be related to the solution reflected in the diagram (constructed without the final developers’ opinion), or to the low traceability of the diagram with the source code.

- There is also a problem related to the nature of the diagram. The developers believed that diagrams should be mainly a representation of the technical layer. But some architects have a tendency to move towards business orientation (modelling business process and business concepts). Furthermore, in the business, representatives interpret UML diagrams as technical pictures.

- There is a big gap too between the source code of a system and its documentation. This is based on the fact that when time pressure appears in a software (maintenance) project (which is very common) one of the first things that is sacrificed is the documentation. So, at some point, the documentation becomes out of date and no longer represents the system, which means that maintainers can no longer trust the documentation.

6.5. RECOMMENDATIONS

Using the results of the case study presented in this paper as a basis, we would like to provide the people involved in modelling and documenting practices with some recommendations. Some of them are already being considered by the interviewees and they are mentioned as best practices; others are “wishes”. We summarise those recommendations that we considered to be most noteworthy in the following list, organised by categories. This list of recommendations contributes to answering the RQ4.

6.5.1. Purpose of use

- Reflect on and define the purpose of using UML [Int24], and also define which diagrams are going to be used for each purpose.
- Involve developers in the software design process [Int4].
- Do not throw documentation ‘over the wall’, but have interactive (face-to-face) meetings where the models are explained and questions can be asked about it. The absence of questions is more likely to be an indication that people did not look at your diagrams. By looking at the results produced by ‘consumers’ of the documentation you can detect whether there are any gaps in the documentation [Int4].
- The architect should send the model to the team, particularly in offshored projects. That team should then review this and provide the architect with feedback to be checked (all using UML) [Int4].
6.5.2. Processes

- Ensure that the documentation gets updated:
  - Plan for the creation and updating of the documentation [Int11]. Reserve time in the project to update documentation [Int11, Int19].
  - Updating models/and documentation should be part of your ‘definition of done’ [Int24].
  - Provide clear criteria that define who updates, as well as when and how documentation should be updated:
    - “The moment you change the structure, you should update the diagram” [Int26]. Or at least, maintain the documentation every week/month [Int4].
    - Define the person/role responsible for maintaining the documentation.
  - Keep a to-do list (backlog) of documentation updates that need to be performed [Int37].
  - Establish a way by which all stakeholders of a project can quickly and easily determine whether a design/document is up-to-date.
    - Establishing a method of versioning may be part of a solution to this.

- With regard to the team size and its relation to documentation:
  - The use of modelling and documentation becomes more useful in the case of teams of 6-8 people or more [Int13, Int28].
    - Documentation is more useful for projects that last longer than 1 year.
  - Decide on a maintenance/migration policy for models (and documentation) [Int24]. When migrating legacy software, pay particular attention to when diagrams/models are introduced. Special guidelines may be useful to standardise which diagrams to introduce and, if making separate additions to existing documentation, where new types of documents are stored in repositories/file systems. [Int18]. Moreover, legacy documentation may not be well structured, and a significant effort may therefore be needed to arrange available information into structures/templates of the target documentation standard [Int18].
  - Finally, in order to improve quality assurance, we recommend the integration of peer-reviews for models and documentation in the development process[Int6]

6.5.3. Training

- Invest in training people on how to use the modelling tools, showing not only their main features but also their details, given that the effective use of UML requires the employment of advanced CASE tools.
- We recommend giving incentives to those software engineers who are not familiar with these advantages and who do not know any possible positive aspect of a change towards using modelling in their process, so that they can become aware of the long-term benefits of using modelling languages (especially UML). Not only this, there is a need for training in that sense. Software engineers should also be encouraged to realise the benefits of maintaining the documentation, and/or be trained in taking advantage of those benefits.
• Teach UML separately from methodology (like RUP). Sometimes it is not easy to apply UML correctly in other methodologies when you learnt the notation as part of a specific methodology. Moreover, some drawbacks of RUP may be projected onto the use of UML.

6.5.4. Standardisation and governance

There is a need for standardisation, which should focus particularly on providing standards/guidelines related to the style of modelling and its archiving:

• Archiving (organisation of files and naming of files).
  o If documents are not used for a long time, people forget where to find them [Int18], and flexible searching mechanisms are therefore important.
• Conventions for naming models, classes, methods and attributes should be defined.
• Define design conventions: which design patterns/strategies are to be used in which situation. The scoping of components and layers should also be established. This helps ensure that similar problems are solved using the same solutions – thus achieving architectural integrity/uniformity.
• Complement the diagrams with personalised legends in order to clarify the non-UML or non-standard parts of the diagram.
• Define the level of detail which should be presented on diagrams. This would help to avoid an excess of documentation, or a lack of it.

It might be helpful to ensure that producers and consumers of documentation know each other. This would thus enable them to agree on standardising the aforementioned issues, and also agree on the relevant information to be documented and the level of abstraction which is needed in each case. Moreover, it will increase compliance if people believe they are ‘helping a colleague’ [Int13].

Alignment of vocabulary is also necessary, but this is independent of the use of UML [Int4]. Yet at the same time, because UML is a standardized notation, UML can help in the definition of the common vocabulary.

Some standardisation is also needed at project and organisational level in order to solve problems related to the accessibility of documentation, always using the same project structures (for example in directories and documents naming, typed of files, etc.). It would also be helpful to use standard templates for documents [Int18].

6.5.5. Tooling

• Make modelling tools available from the start of the project, in order to obtain their benefits from early stages in a project.
• Use of tooling for automated checking of UML, e.g. consistent use of patterns, design principles, naming conventions, or correspondence of source code to design. These may be applicable for long-lived projects that have a high maturity (in order to increase the completeness and correctness of the documentation).
• Use tools that support searches in models.
- Finance tooling centrally – not at the project level. This prevents projects from trying to circumvent modelling by avoiding (generally small) tooling costs.
- Use tools with functionalities related to the traceability between model and text so as not to leave the documentation and model out-of-synch.
- A very important issue which should be improved is the need to keep diagrams and the documentation in-synch with source code, representing all the changes made to the system in them. In order to keep the diagrams updated, we recommend the use of a version management tool of diagrams.

### 6.6. SUMMARY OF RESULTS BY RESEARCH QUESTION

The main objectives of this case study were formulated through research questions, whose answers are summarised below:

**RQ1) What practices are involved in using UML in software maintenance projects?**

We found a wide variety of practices of UML modelling across different projects in the case company. Projects report that the main purposes of using UML are for ‘communication’ and ‘getting an overview’. The purposes mentioned next most frequently are ‘creating a design’ and ‘own understanding’. This is in line with other surveys (Petre, 2013) about UML and highlights the role of UML as a ‘boundary object’—i.e. as a representation of project knowledge used by different stakeholders in different ways. We confirm previous studies which have found that UML modelling is used in a quite loose manner. In our study, we explain some factors that drive the use of UML to a less formal level, which are summarised below:

- Models in project documentation should follow the ‘least common denominator’; i.e. they should be understandable to all audiences of the documentation – also those that have no formal training in UML (such as the project manager or stakeholders from business units).
- Software modellers are conservative as regards the use of detailed features of UML syntax, which is prone to change over versions of the UML standard. They prefer to be a little more high-level rather than to need to update the models when new versions of the UML standard appear.
- The main purpose of UML models is communication, and in particular communication of an ‘overview of the system’ and of ‘design intent’ – rather than ‘design blueprint’. These considerations drive models to focus on key parts of the design. As a consequence, these overview-diagrams intentionally abstract from details. Moreover, in order to convey knowledge/ideas in the best way possible, producers of documentation want to have freedom in which graphical elements to use and freedom to combine freeform text with diagrams. Current UML tools provide very poor support for these combinations.

Overall, because there are multiple stakeholders that use UML models for different purposes, no single perfect UML model exists that is ideal for all. Instead, modelling tools should start catering for different tailorable views for different stakeholders.
With modern software technology, it should be feasible to cater for diverse needs, but the current generation of UML modelling tools does not yet support this.

Some interviewees have asked for executable models or models that support animation. Their claim is that these would be even better for understanding, communication and improving pre-implementation design, but this approach would require a higher level of completeness and detail, and thus incur greater costs. Moreover, it would be more difficult to maintain these models synchronized with the code. In short, the costs-benefits trade-off for executable modelling is unclear.

By and large, reverse engineering is not considered a viable alternative for extracting documentation or models from source code. The benefits claimed for reverse engineering include: i) one does not need to create documentation because one can generate it (i.e. savings of effort), and ii) documentation is always up to date. With regard to argument i): there is ample experience that tells us that reverse engineering cannot be fully automated. Even if some steps can be automated (e.g. identifying all classes and their relations), a lot of manual effort is needed to recover key concepts and abstractions. Furthermore, for some concepts (e.g. sequence diagrams) it is impractical to reconstruct them from the source code because their implementation is scattered across many places in the source code. Moreover, the problems of reverse engineering grow with the size of systems. We therefore found no evidence that savings in effort or development time are made by using reverse engineering models.

Seventy percent of the developers interviewed prefer FD models to RE ones. The situation in which the use of reverse engineering may be practical is that in which programmers need to understand a relatively small piece of the source code that they are working with. In addition to the problem of reverse engineering design models, other concepts are virtually impossible to extract/recover at all from the source code: business processes, design principles, design rationale. Key aspects of such knowledge can thus be documented better.

Finally, there is a negative impact on knowledge sharing: when using reverse engineering to recreate a design model from the implementation (i.e. after the design and programming has been done), this may result in the omission of discussions about the design that would otherwise increase the shared understanding of the design of the systems amongst the developers- i.e. risks of miscommunication is prolonged for a longer period in a project.

RQ2) What are the costs and benefits of using UML in software maintenance projects?

The main cost-factors mentioned for the use of UML actually relate to the change of existing work practices towards using UML, rather than to the actual use of UML. These factors include: cost of training, cost of migrating documentation, and cost of changing processes.

When looking at the actual cost of using UML, the factor mentioned most is that of tooling. With regard to this factor, the quantitative data we elicited at the company
contradicts that tooling is a major cost – e.g. when compared to the costs of training. The costs of tools is therefore perceived to be a high cost factor for large companies, but this is not necessarily so.

It is also interesting to note that there is a radical difference with industries like integrated-circuit (IC) design where design tools can cost up to US $100,000 per year. We think a key difference here is in the purpose: in IC-design the model is a detailed blueprint of the actual IC (hence an engineering construct), whereas in software design, the main purpose of the model is communication. For software modelling, the cost of tooling is perceived as a hurdle, while it is not really so much so in practice.

One factor that complicates the business-case for using modelling is the fact that the benefits are difficult to quantify (they are ‘intangible’): no company keeps a record of miscommunication, poor early design choices, out-of-date or unavailable documentation and their associated costs. One of the reasons why such benefits are hard to quantify is that modelling is often one of many ways (employed in conjunction with others) of achieving a particular goal. Faults cannot therefore be traced back to a single cause.

On the benefit side, software engineers from our case study indicate that the use of UML modelling contributes the following benefits:

1. Process benefits:
   - improved communication / fewer misunderstandings – especially across organisational and geographic boundaries (global software development and outsourcing).
   - It helps to improve the design before implementation (through increased ease of peer-reviewing).
   - It prevents knowledge evaporation.
   - It makes diagnosing of problems easier (especially behaviour models).

2. Product quality benefits
   - 89% of the engineers believe that the use of UML improves the quality of the ultimate software product. More specific findings mentioned in support of this are that UML modelling:
     - It increases the understanding of the system to be built.
     - It enables them to monitor whether an implementation conforms to a design.

   In general, structural models like class diagrams and component diagrams are not believed to make strong contributions to preventing programming defects in the source code, but they are thought to have a positive impact on the structure of the system (modularity, layering). Modelling therefore makes a better contribution to maintaining a good structure of the system. And this is known to benefit the maintainability of systems. What is more, behavioural models (sequence diagrams) share most of the generic benefits of modelling, but also aid in the diagnosis of errors.

   While all of these are mentioned as benefits, there is no empirical evidence regarding the magnitude of their impact on a project for any of them.
RQ3) What are the hurdles when maintaining documentation, and UML models as part of it?

Here we summarise the conclusions obtained as regards both maintaining documentation and using UML as part of the documentation.

**Maintaining documentation**

A common driver for skipping both creating and updating documentation is time-pressure. Unfortunately projects always have time pressure and there is a lack of evidence to support the benefits of having documentation. Moreover, there is currently a normative challenge in that people (mis)interpret the guidelines of the Agile Manifesto (‘working software over comprehensive documentation’) in such a way that only working source code is important, and no documentation is needed (‘the source code is the documentation’).

Other practical hurdles in our case were: the fact that various duplicates of documentation were stored in different archiving systems. This complicated the finding of documentation, as well as the verification of it being up-to-date.

A practical hurdle is that there is often no clear definition of who is responsible for updating documentation. Who should update and when to do so, is often not formally embedded in the development process and there is no quality assurance on updates.

Another hurdle found is related to the misalignment of the incentives for maintaining the documentation:

1. Knowledgeable/experienced developers are required to create documentation, but these are not the people that benefit from it. It is the newcomers and inexperience project members that benefit from the documentation.
2. From a project management perspective, there is also a short-term versus a long-term trade-off. Investing in a good design may lead to easier/cheaper maintenance in the long run. Incentives here are often also misaligned. This could be because the party/part of the organisation that pays for development is often not the same as that which pays for maintenance.
3. The third misalignment is that between those who produce documentation and those who consume documentation. Generally speaking, documentation is created by engineers who have experience with the system they are documenting. This is necessary because they know what the key knowledge about the system that needs to be documented is. But experienced developers do not need the documentation themselves. They produce this for engineers that are relative ‘novices’ in the system.

**Using UML in documentation**

One hurdle as regards using UML was mentioned in response to the first research question: it is the hurdle of introducing/migrating to a practice of UML modelling.

Some people/projects that are modelling find it difficult to find a proper level of abstraction and level of detail for using UML models in their documentation. This can be addressed by having company standards and examples that prescribe this. Our
study also shows that different stakeholders have different preferences for viewing – including different LoD – the documentation of a project.

Another hurdle is the difficulty of keeping UML and implementation synchronized. UML models in documents are often represented at a medium to high level of abstraction, thus leaving out implementation level information. This choice seems wise from the perspective that this requires few updates to the UML models if minor changes are made to the source code. The downsides are that: i) programmers cannot find detailed guidance in the UML models, and ii) it is unclear which changes to the source code need to be reflected in the UML models.

Given that UML models are primarily used for communication, people not trained in software engineering find the UML notation difficult to understand. Any team will need to find a ‘lowest common denominator’ for the notation that they use.

**RQ4) What are best practices when using diagramming and modelling in documentation?**

In our study, the detailed recommendations about best practices can be found in Section 6.6.5, but we highlight the key practices here:

- **PURPOSE OF USE:** Find out which stakeholders in a project use UML/documentation and for what purpose. Then have both the parties that produce documentation and those that consume documentation agree on the level of abstraction and level of detail of UML/documentation.

- **PROCESS:** Clearly define the responsibility for updating/maintaining the (UML) documentation. It should be clear who needs to do this, when this needs to be done, and how. Such practices can be embedded in modern agile approaches – for example, by including updated documentation in the ‘definition of done’.

- **TOOLING:** The archiving of documentation and models should be handled such that it becomes i) easy to find, ii) easy to search within, iii) easy to see if it is up-to-date, and iv) easy to navigate between documents.

- **TRAINING:** Incentivise and/or train in the long term benefits of using modelling languages (especially UML) in the case of those software engineers who are not familiar with these benefits and who do not know any possible benefit of a change towards using modelling in their process. It would also be important to train maintainers on how to use modelling/CASE tools.

- **STANDARISATION AND GOVERNANCE:** Tooling, training and standardization should be managed at a central level so as to achieve uniform practices across projects and to avoid ineffective attempts at cost savings. Moreover, processes and incentives that ensure that the processes and standards are actually followed need to be instated. Organisations should also create a culture that has a realistic understanding of the value of models and documentation and which thus neither overestimates nor underestimates its value.
6.7. THREATS TO VALIDITY

We must consider certain issues which may threaten the validity of the case study (Runeson et al., 2012). In this section, we therefore discuss the threats to the validity of this study. These threats to validity will be presented in the order of their importance (Wohlin et al., 2000): internal validity, external validity, construct validity, and conclusion validity.

- **Internal validity**: The main threat to the internal validity of this study concerns our ability to control influences from other factors beyond those which have been accounted for in this study. For example, the age of the interviewees, the relationship of the interviewees with their team members, or their motivation, might be influential factors as regards being for, or against, the use of UML. Also the consideration of the term UML as a synonym of Rational Unified Process or even Object Orientation might be an internal threat to validity to this study.

- **External validity**: this concerns limitations as regards generalising the results of a study to a broader industrial practice. The sample of the case study and interviewees might be a threat to the validity of this study, although the sampling process was as randomised as possible. We acknowledge the fact that using only one case study may limit the generalisability of the results of this study. However, we believe that reporting these early findings is necessary, as it serves as an encouragement for other researchers to replicate our study using different case studies. The generalisation of the results might be extended to cases which have common characteristics. On the other hand, an interview provides “spontaneous recall” of an answer if it lists a concrete example-instance on an open question: “Q: For example, which diagrams do you use?” “A: Class diagrams.” If in a subsequent question, we ask a closed question: “Q: Do you also use sequence diagrams?” “A: sometimes, but not always.” Then this illustrates that the person actually does also use some other diagram, but needs to be triggered/queued to say so. In interviews it is not always possible to get the interviewee to recall the exact information that is relevant to the question/research. From this perspective, an interview study should not be considered to be complete or accurate in a quantitative sense. Nevertheless, quantitative analyses sometimes provide clear indications of trends.

- **Construct validity**: in the data collection multiple sources of evidence were used. Also the transcript of interviews and observations were sent back to the interviewees to enable correction of raw data. In addition, analyses were presented to them and to the internal research supervisor, in order to maintain their trust in the research. The validity of the developed theory would need to be tested in other case studies.

- **Conclusion validity or reliability**: this relates to the ability to draw a correct conclusion from a study. The chain of evidence from the interviews and documentation analysed through to the synthesized evidence was maintained using a word-for-word transcription. This analysis took a long time to carry out, but this
was due in part to our desire to ensure that we did not make mistakes in the interpretation while the analysis was being undertaken. We therefore asked the interviewees to give feedback to the researchers on the transcripts of the interviews. This practice is known as ‘member checking,’ and it was used continuously to obtain feedback on both the transcripts and the analyses. Tools were also used during the analysis of the data. Furthermore, the individual coding performed by each researcher was discussed by them, so that they could verify and reach an agreement on them. In particular, we used triangulation in order to reduce bias. Triangulation (Robson, 2002) refers to having multiple sources for the study information. In this study, this was attained in three different ways, which further increases the validity of the study. A summary is provided below:

- Data triangulation: Multiple data sources were used in the study, such as interviews with people who had different roles, experience in ICT, etc.
- Investigator triangulation: Interviews were performed by one researcher, but their analysis was done by two researchers together. Important analysis steps were performed by two researchers independently.
- Methodological triangulation: Multiple methods were used; both qualitative interviews and qualitative archival analysis were employed, along with a few quantitative measures to investigate relevant metrics.

### 6.8. CONCLUSIONS

In this paper, we have analysed the practices and use of software modelling for software maintenance in industry. We pay particular attention to how UML is being used as part of the documentation available in a maintenance project. This analysis has been carried out by means of a case study involving 31 interviewees playing different roles in a variety of projects in a software department at a multinational company. The qualitative data obtained is presented in conjunction with a theory about how some elements from a common SE approach (like goals, process, practice and tools) influence the documentation or modelling approaches.

The majority of the interviewees consider that diagrams are a helpful tool in software maintenance. The most commonly-mentioned purpose of use of diagramming software designs is ‘overview’, especially of spatial structures with many components and their mutual relations. Moreover, a graphical notation is an aid to achieving more uniformity of documentation in comparison to textual documentation. It was also noted that diagrams are easy to compare (to other diagrams). Engineers find it easy to judge the relevance of diagrams for their information need, and can therefore quickly glance over diagrams in search of certain pieces of information.

We also detected that the richness of syntactical elements in a notation (like UML) may lead to discussions that do not add any value (‘which type of rectangle to use’). This is hypothesised on the basis of discussions about colours in layout that were perceived to be a waste of time.
Moreover, some respondents believe that UML training is a one-time investment (training in UML is needed only once). Other interviewees, however, believe that it is important to have refresher courses in UML, especially for people that use UML occasionally (at a low frequency; several times per year). This would keep them informed about the latest developments in the notation. This instruction could also be triggered by publications of a new version of the UML standard or a significant new release of the tool.

It is probably for this reason that software modellers are conservative in using detailed features of UML syntax that is prone to change in versions of the UML standard. They prefer to be a little more high level rather than needing to update the models when new versions of the UML standard appear. A list of best practices was also extracted from this case study; some of them are based on practices already applied by the interviewees and considered by them as recommendations to people involved in software maintenance projects (or in software modelling), and other are their “wishes”. We have classified this list of best practices in five categories: tooling, training, purpose of use, process and standardisation and governance.

Overall, the engineers at the company have a positive attitude towards the use of graphical modelling like UML in software maintenance. Moreover, the discussion should not be framed as ‘black-or-white’, but as a search for a practical way in which to capture and share knowledge about a system, which will inevitably be a combination of text and diagrams. While quantitative evidence on the pros and cons remains elusive, we believe that the views of the engineers in this company represent a large body of experience.

6.9. FUTURE WORK

Several recent empirical studies into the use of modelling have claimed to be representative of the community of professional software engineers. Our case study highlights that there are many different types of roles (all of them software professionals) in large software development and maintenance projects. However, the involvement of different roles with modelling varies widely. Hence, for future studies, we recommend that researchers pay special attention during the selection of participants in their studies (be they interviews or surveys or experiments), and also in the analysis of their data split, in an attempt to see whether patterns emerge if participants are grouped based on their role. It is not sufficiently detailed to refer to general roles such as ‘software engineers’ or ‘software professionals’.

Possible future directions for our current research are related to the presentation of certain topics which were detected as candidates during the current research, such as:

- Does (just enough) up-front design indeed prevent downstream repair, and to what degree?
- One big question seems undecided: Is the updating of models/documentation time-consuming or not? Or, in other words, how much time do software architects/designers spend on creating and updating models?
• There is a fundamental problem as regards dividing the documentation in a proper way for the purpose of presenting different views to different stakeholders. It might be interesting to research what the proper views for this purpose are.

• Several participants reported that the benefits of creating designs/models are hard to quantify. This area needs new creative approaches in order to attain more insights into this matter.

Our study also identified several problematic areas in current tool support for modelling during software development. There is ongoing research that addresses some of these issues, such as merging and differencing, versioning of models, etc. One issue that stands out is the lack of support for the flexible mixing of diagrams and text. Another issue is the need to abstract models from source code at different levels of abstraction.

Acknowledgements: We are very grateful to the company for dedicating time to us and opening up to us in interviews.

This research has been funded by the SEQUOIA project (Ministerio de Economía y Competitividad), and by the Fondo Europeo de Desarrollo Regional FEDER, TIN2012-37493-C03-01.
CHAPTER 7.  CONCLUSIONS
Chapter 7: Conclusions

This chapter presents the conclusions of this PhD Thesis, and is organized as follows: Section 7.1 provides a summary of the main findings obtained after carrying out the research developed during this PhD Thesis. The initial research questions and the achievement of the proposed research goals are also analyzed. Section 7.2 outlines the main contributions of this PhD Thesis and its implication for academia and industry, while Section 7.3 states some research lines that are pending for further research. Finally, Section 7.4 lists the whole set of papers published throughout the realization of the PhD Thesis.

7.1. ACHIEVEMENT OF GOALS AND SUMMARY OF FINDINGS

This section reviews the goals established in the Introduction of this PhD Thesis and analyses the way in which they have been achieved.

The main objective of this PhD Thesis was to answer the research questions explained in the Introduction chapter. The following sub-sections show how each Research Question was answered.

7.1.1. RQ1. What is the perception of professionals in industry as regards the value of using UML in software maintenance?

This question helped to provide maintainers’ real opinions as regards how UML diagrams (and documentation in general) enable them to perform their maintenance tasks, among other issues. It also helped to establish what kind of companies are fruitfully using UML diagrams as part of their software documentation.

We performed a survey with 178 software maintainers from 12 countries, presented in Chapter 5. The results of the survey revealed some remarkable software maintenance practices, which are summarized in the following paragraphs.

The use of a graphical notation as a complement when attempting to understand the system that will be maintained and to support the design of the changes related to software maintenance tasks is widespread. When a graphical notation is used, the notation selected in the majority of the cases is UML.

As expected, the UML diagrams that are used most frequently during software maintenance are class diagrams, use case diagrams, sequence diagrams and activity diagrams.

Maintainers do not always use the available documentation (with or without diagrams), and use only the source code and its comments. This is owing to problems concerning a lack of synchronization caused by non-updated documentation/diagrams.

If we focus on issues regarding the process, it will be observed that the origin of these UML diagrams is almost always the development phase; i.e., they are not created especially during the maintenance, except when UML diagrams from the software development are not available (in which case they are created specifically during maintenance). Performing reverse engineering techniques is difficult, despite the fact that there are automatic reverse engineering tools. But after the automatic
process, a manual process is required during which the architect attempts to “clean” the diagrams.

- It was detected that some experience is needed in the ICT field in general before working in the software maintenance field. This may be owing to the need to have sufficient experience as regards understanding how systems are built before being able to modify them (see Figure 7.1).

![Diagram](image)

**Figure 7.1. Contextual factors that affect the adoption of UML diagrams.**

- It is also important to note that most UML users are highly-educated and experienced. It would appear that a higher educational level leads to an increased (likelihood of) use of UML diagrams.
- The size of the system being maintained seems to be an influential factor, because the results obtained show that UML diagrams are extremely popular when the team has to maintain a very large system. This would appear to be logical, since one of the reasons put forward for using UML (or models) is that it helps manage large and/or complicated systems.
- The size of the maintenance team also appears to influence the use of UML. Larger teams use UML diagrams more frequently, proportionately, perhaps because of the improvement to the understanding of the system provided by UML diagrams and owing to the need to share/communicate knowledge in this kind of teams. This is also because small teams use fewer UML diagrams than large ones, since they have facilities for face-to-face meetings and, therefore, require less supporting documentation. It is important to note that larger teams usually work on large (and thus complex) systems, signifying that there is more need for abstraction/managing complexity, and UML assists in this respect.
The survey also highlighted several perceived benefits of using UML during software maintenance, which are related to modeling purposes:

- The main reason for using UML is that less time is needed for a better understanding of the system being maintained; this improves the detection of defects.
- Moreover, when UML diagrams are available as part of the documentation, less effort is required to consult them.

If we focus on the types of tools used to manage (read or modify) UML diagrams, it is quite surprising that the UML Case tools and non-specific UML design tools are used to the same extent. One reason for the frequent use of a non-specific tool might be that a lot of companies work with it for other purposes; employees already have the tool installed, signifying that it is easier to use it and to share diagrams (although they do not check the correctness of the syntax of the diagrams). It is also important to note that some UML users do not use a modeling tool because they manage diagrams on physical paper or blackboards, in which case the diagrams are not digitalized or they are stored as non-modifiable images. The variety of practices related to what tool is used could be a reflection of the fact that different tools are used for different needs: for example, a UML CASE tool could be used to check syntax or to attain a higher traceability diagram-code, while a non-specific tool could be used for a quick brainstorming.

7.1.2. RQ2. Does the level of detail (LoD) of UML diagrams influence software maintenance?

As stated in the introduction, it would also be desirable to help organizations to define the proper LoD required to update the UML documentation in order for it to be synchronized with the source code and thus attain understandability benefits during further maintenance. In order to help organizations to establish a baseline LoD for their UML diagrams, we carried out a family of experiments with 81 students in order to discover the appropriate LoD for UML diagrams. This family of experiments is presented in Chapter 3.

We found that the preferred LoD depends on the purpose of the diagram: there is a slight tendency in favor of using high LoD diagrams to understand the system, and a low LoD to maintain it.

Although high LoD diagrams are preferred when understanding a system that is being maintained, the participants stated that they had more difficulties when reading high LoD diagrams as compared to low LoD diagrams. This can be explained by the fact that the structural complexity of a software diagram affects its cognitive complexity and maintainers, therefore, like a higher LoD because it provides them with more detailed information about the actual implementation details. But the higher amount of (implementation) details also makes the model more difficult to understand.

Despite the fact that the participants stated that they experience more difficulties when reading high LoD diagrams, the majority of them considered that the presence of
some elements represented in the high LoD versions of each diagram is very valuable: attributes and operations in class diagrams and formal messages with parameters in sequence diagrams.

When the domain is well known, or the maintainers are already familiarized with it, this minimizes the time taken to read the UML diagrams and they may work directly on the source code. In conclusion, when the system maintained is small and the application domain is simple and is a well-known domain, then novice software maintainers do not effectively require the use of UML diagrams or other types of technical documentation, independently of their LoD, in the context of maintenance projects for Java systems. We therefore assume that, in this context, the use of UML diagrams does not increase the performance of maintenance operations and might even distract the participants while performing comprehension and modification tasks.

In other contexts, such as when the systems maintained are large and/or technically complex, novice maintainers prefer to use the high level design diagrams/documentation. They use the diagrams in order to roughly understand the system architecture and thus which components/subsystem they have to understand and modify. Then, when the system is understood, they prefer to use low LoD diagrams when performing the maintenance tasks.

Although the results of this family of experiments are encompassed in the case of novice maintainers when they are modifying small and well-known domains, we assume that understanding is also improved in medium to large systems and/or those from unfamiliar domains because they are the environments that benefit most from the presence of a proper (UML) design. However, the investment in the maintenance of the UML documentation required for an appropriate maintenance of a system should not be high.

7.1.3. RQ3. What is the impact of using UML in software maintenance in terms of its use in a software project?

There is not much empirical evidence on the impact of UML modeling in software maintenance in an industrial context. It is sometimes difficult to measure the effect of the UML notation in a real software project, in which there are many uncontrolled variables that might influence the results. This led us to make our first attempt to obtain an industrial point of view concerning the influence of UML on maintenance tasks (for example, the perceived benefits and hurdles of UML modeling practices) through the use of a case study (see Chapter 6) in order to. The data used in the case study was collected at the ICT Department of a multinational transport company in Western Europe. This company is about 100 years old, but we were able to trace the use of ICT in the company back to (at least) the late 1960’s. The ICT department has between 800-1000 employees, and plays an essential role in this company’s competitive advantage within its sector. This provides an industrial point of view concerning the influence of UML on maintenance tasks.

Firstly, during this case study we detected that UML modeling is commonly used at a low level of formality. The main reason for this is that the purpose of UML
models is communication and in particular the communication of an “overview of the system”, which drives users to focus on key parts of the design. Moreover, models in project documentation should be understandable for all the audiences of the documentation.

After detecting why UML is used, we focused on the perceived costs of modeling. The main cost-factors mentioned as regards the use of UML relate to the changing and migration of existing work practices towards using UML, rather than to the actual use of UML. These factors include: the cost of training, the cost of migrating documentation and the cost of changing processes. What is more, the cost of tools is perceived to be a high cost factor for large companies, but this perception is not necessarily so. With regard to the costs, we can also consider some of the hurdles detected as regards the adoption of UML modeling, from the general to the specific. It is first necessary to highlight some hurdles as regards maintaining documentation by itself (non UML).

The hurdles when maintaining UML as part the documentation are the following:

- The most frequently mentioned hurdle was time-pressure, but unfortunately projects always have time pressure so this should not be used as an excuse and projects need to reserve time for a proper management of the modeling.
- Another hurdle is that there is often no clear definition of who is responsible for updating documentation. The updating is further complicated by the fact that duplicates of documentation have sometimes been stored in different archiving systems.
- The difficulty of introducing/migrating to a UML modeling practice was also mentioned, but this is not a specific hurdle of UML and occurs with any migration approach.

These factors make the use of (UML) documentation unnecessarily complicated to manage.

We shall now focus on the hurdles that are directly related to UML modeling alone:

- It is difficult to find a proper level of abstraction and level of detail in order to use UML models in the documentation.
- The difficulty of keeping UML models and the implementation synchronized was also frequently mentioned.

Although the current generation of UML modeling tools does not yet support all these features, it is technically possible to develop a tool with which to solve the majority of the aforementioned hurdles.

If we now concentrate on the beneficial aspects, one perceived benefit is a reduction in the execution time of a project. Some process benefits were also highlighted, such as: improved communication, the prevention of knowledge evaporation and the easing of the diagnosis of problems (especially when using behavior models). UML modeling is also considered helpful as regards improving the
design before starting implementation activities (through an increased ease of peer-reviewing).

If we focus on the product quality benefits of using UML modeling, the majority of maintainers believe that it improves the quality of the ultimate software product. This increased quality might be the result of some of the other benefits mentioned, such as an increased understanding of the system to be maintained and the possibility of monitoring whether an implementation conforms to a design. Finally, in general, structural models like class- and component-diagrams are believed to have a positive impact on the (quality of the) structure of the system (modularity, layering). Modeling, therefore, contributes to the achievement of a good system structure, and a good system structure is known to be a major benefit as regards the maintainability of systems. What is more, behavioral models (sequence diagrams) share most of the generic benefits of modeling, but also aid in the diagnosis of errors.

Finally, we should stress that the benefits and costs/hurdles are difficult to quantify because they are intangible: no company keeps a record of miscommunication, poor early design choices, out-of-date or unavailable documentation and their associated costs. One of the reasons why such benefits are hard to quantify is that modeling is often one of many ways (employed in conjunction with others) of achieving a particular goal. Faults cannot, therefore, then be traced back to a single cause.

7.1.4. **RQ4. Are forward designed or reverse-engineered UML diagrams more helpful for code maintenance?**

Projects sometimes use Reverse Engineering (RE) when there is no Forward Designed (FD) documentation available, or as a way around having to spend time on creating FD diagrams. In order to discover what is more beneficial for software maintainers (RE or FD diagrams), a family of experiments with in total 169 students was performed (see Chapter 4). The results obtained revealed that:

- Software maintainers were a little more efficient and effective when using FD UML diagrams that were made by hand during the design phase. This indicates that FD diagrams may, to some extent, improve the maintenance of source code.
- Only in the case of junior maintainers was a better efficiency obtained when using the RE diagrams. This might be explained by the fact that RE diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams.
- In general, those who received FD diagrams experienced fewer difficulties when reading the diagrams when compared to those who received the RE diagrams. RE sequence diagrams were simultaneously found to be less useful than FD sequence diagrams. These participants pointed out that these RE sequence diagrams are not very useful owing to their low level of readability/high level of details. A possible explanation for this is that human engineers make (implicit) assessments regarding which information is important in order to understand key aspects of the system and thus regarding which information should be included in a diagram. We believe that the findings of our experiments indicate that FD class diagrams provide a more
attractive balance between detail and relevant information than RE class diagrams. But, as stated previously, UML diagrams are not usually updated during maintenance tasks owing to the time constraints in industrial environments. The high level use of class diagrams observed during this family of experiments therefore leads us to recommend that companies keep UML diagrams up to date in order to improve maintainers’ performance. Sequence diagrams are used less often than class diagrams, probably because of the nature of the tasks (perfective maintenance tasks), and because it would appear that current automated reverse engineering methods are not able to filter out the relevant information in a sequence diagram when employed for maintenance tasks. These factors lead maintainers who are using RE diagrams to work directly with source code.

- Class diagrams are important artifacts that are widely used and highly appreciated by maintainers. However, there is a lack of clarity as to whether the documentation, and UML as part of it, is up-to-date. This goes against an effective use of the diagrams.

7.2. CONTRIBUTIONS OF THE PHD THESIS

The main contribution of this PhD Thesis is the enrichment of the SE body of knowledge concerning the impacts of UML modeling on software maintenance. The different empirical studies performed provide different points of view (academic lab, industry, and company) regarding the factors that influence the use of UML and its impact on software maintenance. The main factors studied in each empirical study (survey, experiments and case study), and the main relations among them are summarized in Figure 7.2. In particular, the realization of this empirical research has allowed us to clarify common beliefs or theoretical assumptions related to the impacts of UML modeling by assessing those assumptions in academic contexts and in different projects in a real industrial context.

Figure 7.2. Main factors studied and relations among them.
In essence, the contribution of this PhD Thesis is two-fold. First, this study provides sound empirical evidence about the potential benefits of UML modeling in software maintenance. The findings of this study may also contribute to the body of knowledge, particularly in the field of software engineering. Additionally, and from a research perspective, we consider that this study is a milestone towards a more comprehensive understanding of the role of modeling in software maintenance.

The knowledge obtained throughout the realization of this PhD Thesis and its practical implications are summarized below:

- We detected that following a modeling approach, and especially UML, might be part of the approach to documentation, and also part of the approach to implementation. But going further, the UML modeling approach could be considered as a “bridge” between documentation and implementation. In addition, the elements of this UML modeling approach which were influenced by the decisions made as part of the software engineering approach have been highlighted (Figure 7.3). Although scholars have, in some respects, studied the use of modeling notations within software maintenance teams, our knowledge remains fragmented by the divergent efforts that are based on and contribute to theoretical perspectives. This PhD Thesis contributes with a more concrete and less fragmented theory regarding the use of UML.

- It is very important to target the appropriate level of detail in UML models according to their purposes. In particular, UML models that serve as guides for implementation usually require a high LoD. This high level would be beneficial for those maintainers who are not involved in the development of a specific system and who need to understand it in order to maintain it. When the system is already...
understood, a lower LoD is recommended, when UML diagrams are used to perform maintenance tasks or when communicating design decisions. But it is essential that the UML documentation be synchronized with the source code, even though this may be achieved by means of an abstraction step.

- The use of low LoD diagrams is better in comparison to that of high LoD diagrams when performing maintenance tasks (once the system being maintained is understood). This statement is also supported by Briand et al.’s framework (Briand et al., 2001b), which hypothesizes that high cognitive complexity (i.e., high LoD diagrams) will result in reduced understandability which impedes the analyzability, adaptability and flexibility of the diagram. It is useless to invest too much in software modeling so as to provide maintainers with very detailed information.

- In order to achieve quality and productivity improvements through the use of UML in software maintenance, we recommend enforcing the practice of creating UML diagrams during the software development, and maintaining them (and the rest of the documentation) in correspondence with the source code. When UML diagrams from the development phase are not available, we recommend generating them using a Reverse Engineering technique. But we also would recommend revising those RE UML diagrams in order to clean and clarify them for a proper understanding.

- Since two key purposes of UML models are communication and attaining an overview of the system, both the parties that produce documentation and those that consume documentation should agree on a level of abstraction and level of detail of UML/documentation that best suits all parties. All parties should be involved in the creation of the UML models and have interactive meetings in order to avoid any possible misunderstandings about their content.

- In software maintenance projects it is necessary to clearly define the responsibility regarding updating/maintaining the (UML) documentation. It should be clear who needs to do this, when this needs to be done, and how. Such practices can be embedded in modern agile approaches – by, for example, including updated documentation in the ‘definition of done’. Establishing a versioning method would also be desirable.

The second contribution of this PhD Thesis is related to modeling practices. This PhD thesis provides recommendations concerning best modeling practices when using UML. These recommendations are based on both the empirical data obtained from an industrial case study and expert opinions. This study, therefore, essentially serves as a guide and as a justification for software engineers to continue using (and updating) the UML models of software systems, and to obtain the maximum benefit from them. A list of our main recommendations is summarized below:

- We recommend that efforts should be made to design the UML models during the software development phase in order to have all the benefits of the modeling available during maintenance (apart from the benefits that could be obtained in the development). Following a model-centric approach as one of the companies’ software practices improves the understanding of the system and source code.
• The use of modeling and documentation becomes more useful in the case of teams of 6-8 people or more, and also for projects that last longer than 1 year.

• In order to obtain the aforementioned paybacks in the maintenance phase, during the development of a system it is important to:
  o Reflect on and define the purpose of using UML, and also define which diagrams are going to be used for each purpose.
  o Involve developers in the software design process.
  o Do not throw documentation ‘over the wall’, but have interactive (face-to-face) meetings in which the models are explained and questions can be asked about it.
  o The architect should send the model to the team, particularly in offshored projects. That team should then review this and provide the architect with feedback to be checked (all using UML).
  o It is also important not to construct overly long documents or unstructured documentation. This will help maintainers to search for pieces of relevant information.

• In an ideal software project, the development of a software system begins by creating UML diagrams, in addition to keeping them up-to-date during the maintenance, thereby making it easier to perform maintenance tasks. As mentioned previously, it is recommendable to maintain the documentation, and especially the diagrams, in order to obtain all their possible paybacks, even when the system being maintained is well known. During the maintenance it is important to:
  o Ensure that the documentation gets updated. We recommend the use of a diagram-version management tool and taking care when planning for the creation and updating of the documentation, reserving time in the project to update documentation. It is also very important to provide clear criteria that define who should update the documentation, along with when and how it should be updated.
  o Decide on a maintenance/migration policy for models (and documentation). When migrating legacy software, pay particular attention to when diagrams/models are introduced. Special guidelines may be useful to standardize which diagrams to introduce and, if making separate additions to existing documentation, where new types of documents are stored in repositories/file systems.

• The following list of recommendations applies to both development and maintenance:
  o Integrate peer-reviews for models and documentation in the development process.
  o Invest in training people how to use the modeling tools, showing not only their main features but also their details.
  o Train those software engineers who are not familiar with the advantages and who do not know any possible positive aspects of a change towards using modeling in their process, thus enabling them to become aware of the long-term benefits of using modeling languages (especially UML).
Conventions for naming models, classes, methods and attributes should be defined.

Complement the diagrams with personalized legends in order to clarify the non-UML or non-standard parts of the diagram.

Define the level of detail which should be presented in diagrams. This would help to avoid an excess of documentation, or a lack of it.

Ensure that the producers and consumers of documentation know each other. This would thus enable them to agree on standardizing the aforementioned issues, and also to agree on the relevant information to be documented and the level of abstraction which is needed in each case.

Some standardization is also needed at project and organizational level in order to solve problems related to the accessibility of documentation, always using the same project structures.

Use tooling for the automated checking of UML, e.g. a consistent use of patterns, design principles, naming conventions, or correspondence of source code with design. These may be applicable for long-lived projects that have a high maturity (in order to increase the completeness and correctness of the documentation).

Use tools that support searches in models.

Use tools with functionalities related to the traceability between model and text so as not to leave the documentation and model out of synch.

Make modeling tools available from the start of the project in order to obtain their benefits from the early stages in a project.

Finance tooling centrally – not at the project level. This prevents projects from trying to circumvent modeling by avoiding (generally small) tooling costs.

### 7.3. FUTURE RESEARCH LINES

The findings obtained are a first approach to discovering the contextual factors that affect the effective use of UML during software maintenance in industrial projects. We believe that we have now taken the first step, but this topic deserves to be studied in greater depth. Possible future directions for this line of research are:

- It would be interesting to analyze the effect of different UML notations by, for example, comparing formal diagrams with box & lines. Several reasons (such as time pressure or the lack of training on notation) cause maintainers/developers to use informal box and line (UML) diagrams. It would be interesting to investigate the impact of this relaxed use of UML on the understanding and maintenance of a software system.

- Developing supporting tools for a proper UML documentation. The current tools are focused only on managing the UML diagrams, and they include features which help to create correct UML diagrams (such as checking correctness, helping to manage consistency, etc.). But it would be desirable to have a tool with which to create and maintain documentation containing a mix of both text and diagrams. It would be important to have a tool with functionalities that improve the traceability between model and text so as not to leave the documentation and model out of...
synch. It also would be useful to have a tool that supports a correct versioning system of the diagrams, supports searches in models, or a tool capable of presenting different views (e.g. at abstraction levels or in detail) of the same diagram adapted for different purposes (for different consumers of this information).

- An empirical investigation regarding whether this extra effort when creating and maintaining UML documentation is worthwhile is also still pending. This would provide the software community with the knowledge regarding the return on investment of UML modeling in software maintenance. This topic has not been explored in detail, and this is necessary if we wish to increase the acceptation and usage of UML in industry.

- It is important to strengthen empirical validation in order to mitigate the aforementioned threats to the validity of the empirical studies presented in this Thesis. It would, therefore, be interesting to follow the same research line of this thesis and carry out replications of these studies with different configurations or different cases. The families of experiments could be replicated by considering realistic software systems related to larger and unknown domains in order to verify whether the findings obtained are still valid. It would also be interesting to replicate these laboratory studies with industrial practitioners. With regard to the case study, it would be interesting to replicate the research by conducting more industrial studies with different software systems (e.g., different programming languages, technologies, size, complexity, etc.) from different companies, using different software development methodologies or at least those from different domains.

7.4. PUBLICATIONS

As mentioned in the Introduction chapter, this PhD Thesis is presented as a collection of ten papers, published in diverse journals, conferences and books related to Software Engineering.

Other works, apart from the 10 selected to form the chapters of this thesis, have also been published. A complete list of all the papers published throughout the period in which this PhD thesis was being completed is provided below.

Publications in national conferences


**Publications in international conferences/workshops**


**Journal publications (included in the JCR)**


BIBLIOGRAPHY


Oppenheim, A.N. (2000). Questionnaire design, interviewing and attitude measurement (Bloomsbury Academic).


APPENDIX A. LIST OF PRIMARY STUDIES (related to Chapter 2)

The papers considered as primary studies in the systematic mapping study presented in this paper and that have been treated as primary studies are presented below.


Appendix A

of Software Engineering Languages and Technologies (WEASEL'07), pp. 25-30.


APPENDIX B. DEFINITIONS OF MEASURES (related to Chapter 2)

The definition of the measures for the dependent variables used in the empirical studies covered in this systematic mapping study is presented below. In the definition of the measures we use the word question to simplify the definition, but we can also refer to a task. Note that in Table 7, all the studies that use measures which measure the same concept are grouped together (even though they were originally presented with different names), using the names according to the classification set out below.

Correctness
Definition: The percentage of questions that are answered correctly.
Formula: Number of correct answers/Number of questions.
Papers which use this measure: [P9], [P7], [P8], [P9], [P18], [P20], [P10]

Accuracy
Definition: the number of correct answers.
Papers which use this measure with this name: [P23], [P34], [P36], [P37]
Papers which use this measure with different names:
- Total score: [P4], [P21], [P24], [P29]
- Correct interpretation: [P6]
- Number of responses: [P22]
- Comprehension: [P2]
- Without a specific name: [P14], [P17], [P31]

Effectiveness
Definition: The percentage of questions answered which are correct.
Formula: Number of correct answers/Number of answers
Papers which use this measure with this name: [P32]
Papers which use this measure with a different name:
- Correctness: [P7], [P13], [P19]

F-Measure
Definition: It is an aggregate measure which is a standard combination of the recall and precision, defined as their harmonic mean.
Formula: \[ F - Measure = \frac{2 \cdot \text{precision}_{s,i} \cdot \text{recall}_{s,i}}{\text{precision}_{s,i} + \text{recall}_{s,i}} \]
Definition: It measures the fraction of expected items that are in the answer.

Formula: \[
\frac{|A_{s,i} \cap C_i|}{|C_i|}
\]

$A_{s,i}$: Set of elements mentioned in the answer to question $i$ by subject $s$.

$C_i$: The correct set of elements expected for question $i$.

**Precision**

Definition: It measures the fraction of items in the answer that are correct.

Formula: \[
\frac{|A_{s,i} \cap C_i|}{|A_{s,i}|}
\]

$A_{s,i}$: Set of elements mentioned in the answer to question $i$ by subject $s$.

$C_i$: The correct set of elements expected for question $i$.

Papers which use this measure: [P1], [P25], [P26]

**Efficiency**

Definition: The number of correct answers per time units.

Formula: Number of correct answers/Time

Papers which use this measure with this name: [P7], [P32], [P10]

Papers which use this measure with a different name:

- **Efficacy**: [P17]

**Relative time (for a correct answer)**

Definition: It measures the time that a subject took to obtain a correct answer.

Formula: Time/Number of correct answers

Papers which use this measure: [P29], [P7]

**Perceived comprehensibility**

This is a subjective measure obtained as a ranking of the subject’s perceived understandability of a certain diagram. Measured using a 1-5 Likert ordinal scale, where the score of 1 indicated that the diagram was absolutely incomprehensible.

Papers which use this measure: [P3], [P12]

**Perceived ease of construction**

This is a subjective measure obtained as a ranking of the subject’s perceived ease of construction of a certain diagram. It is measured using a 1-5 Likert ordinal scale, where the score of 1 indicates that the diagram is very difficult.

Papers which use this measure: [P14]
**Time**

This is a measure which is used to calculate the number of units of time used to perform a task.

Papers which use this measure: [P3], [P4], [P5], [P7], [P8], [P9], [P11], [P12], [P13], [P14], [P17], [P18], [P19], [P21], [P22], [P23], [P28], [P29], [P32], [P32], [P36], [P37], [P38]

**Errors**

This is a measure which counts the number of mistakes made in solving a specific task.

Papers which use this measure: [P9], [P8], [P9], [P15], [P16], [P27], [P28], [P33]
APPENDIX C. THE SEARCH STRINGS (related to Chapter 2)

The definition of the search strings used in each search engine is presented as follows. As commented on in Table 2.1 we had three major terms, and we also considered alternative spellings and synonyms of, or terms related to, the major terms. The original search string was:

\[(UML \text{ OR (Unified Modelling Language))} \]

\[\text{AND} \]

\[(\text{Maintenance OR Maintainability OR Modularity OR Reusability OR Analyzability OR Changeability OR Evolution OR Evolvability OR Modification OR Stability OR Testability OR Comprehensibility OR Comprehension OR Understandability OR Understanding OR Misinterpretation)}\]

\[\text{AND} \]

\[(\text{Empirical OR Experiment OR Survey OR Case study OR Action research)}\]

Owing to the limitation of the search engines, we observed that such a long string could not be used directly in all the search engines. It was therefore necessary to tailor the search string to each digital library by splitting the original search string and then combining the results manually. The search strings used for each digital source are presented below.

**ACM and IEEE search string**


**Science Direct and SCOPUS search string**

\[\text{TITLE-ABSTR-KEY((UML OR (Unified AND Modeling AND Language)) AND (Maintenance OR maintainability OR modularity OR reusability OR analyzability OR changeability OR evolution OR evolvability OR(modification AND stability) OR testability OR comprehensibility OR comprehension OR understandability OR understanding)} \]

AND (empirical OR experiment OR survey OR (case AND study) OR (action AND research))}
Springerlink search string

The search string was divided into 28 search strings because this string only allows 10 terms to be placed in the search string textbox. After the searches had been carried out, we combined their results using the SLR-Tool, which automatically detects duplicate papers.

String 1: ab:(UML and maintenance and(empirical or experiment or survey or(case and study) or(action and research)))

String 2: ab:(UML and maintainability and(empirical or experiment or survey or(case and study) or(action and research)))

String 3: ab:(UML and modularity and(empirical or experiment or survey or(case and study) or(action and research)))

String 4: ab:(UML and reusability and(empirical or experiment or survey or(case and study) or(action and research)))

String 5: ab:(UML and analyzability and(empirical or experiment or survey or(case and study) or(action and research)))

String 6: ab:(UML and changeability and(empirical or experiment or survey or(case and study) or(action and research)))

String 7: ab:(UML and evolution and(empirical or experiment or survey or(case and study) or(action and research)))

String 8: ab:(UML and evolvability and(empirical or experiment or survey or(case and study) or(action and research)))

String 9: ab:(UML and(modification and stability) and(empirical or experiment or survey or(case and study) or(action and research)))

String 10: ab:(UML and testability and(empirical or experiment or survey or(case and study) or(action and research)))

String 11: ab:(UML and comprehensibility and(empirical or experiment or survey or(case and study) or(action and research)))

String 12: ab:(UML and comprehension and(empirical or experiment or survey or(case and study) or(action and research)))

String 13: ab:(UML and understandability and(empirical or experiment or survey or(case and study) or(action and research)))

String 14: ab:(UML and understanding and(empirical or experiment or survey or(case and study) or(action and research)))

String 15: ab: ("Unified Modeling Language" and Maintenance and(empirical or experiment or survey or(case and study) or(action and research)))

String 16: ab: ("Unified Modeling Language" and maintainability and(empirical or experiment or survey or(case and study) or(action and research)))
Appendix C

**String 17:** ab: ("Unified Modeling Language" and modularity and (empirical or experiment or survey or (case and study) or (action and research)))

**String 18:** ab: ("Unified Modeling Language" and reusability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 19:** ab: ("Unified Modeling Language" and analyzability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 20:** ab: ("Unified Modeling Language" and changeability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 21:** ab: ("Unified Modeling Language" and evolution and (empirical or experiment or survey or (case and study) or (action and research)))

**String 22:** ab: ("Unified Modeling Language" and evolvability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 23:** ab: ("Unified Modeling Language" and (modification and stability) and (empirical or experiment or survey or (case and study) or (action and research)))

**String 24:** ab: ("Unified Modeling Language" and testability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 25:** ab: ("Unified Modeling Language" and comprehensibility and (empirical or experiment or survey or (case and study) or (action and research)))

**String 26:** ab: ("Unified Modeling Language" and comprehension and (empirical or experiment or survey or (case and study) or (action and research)))

**String 27:** ab: ("Unified Modeling Language" and understandability and (empirical or experiment or survey or (case and study) or (action and research)))

**String 28:** ab: ("Unified Modeling Language" and understanding and (empirical or experiment or survey or (case and study) or (action and research)))

**Wiley Inter Science search string**

We used the advanced search in which it is possible to use three (or more) textboxes to enter complex strings (see Figure 1). We used a search string divided into three parts, which were linked by AND connectors. Different textboxes were used to introduce each part of the search string:
UML or (Unified and Modeling and Language)

AND

Maintenance OR maintainability OR modularity OR reusability OR analyzability OR changeability OR evolution OR evolvability OR (modification AND stability) OR testability OR comprehensibility OR comprehension OR understandability OR understanding

AND

empirical OR experiment OR survey OR (case AND study) OR (action AND research)
APPENDIX D. EXAMPLES OF LOW AND HIGH LOD DIAGRAMS (related to Chapter 3)

This appendix shows examples of a low LoD class diagram (Figure D.1) and a high LoD class diagram (Figure D.2).

Figure D.1. Example of a low LoD class diagram.

Figure D.2. Example of a high LoD class diagram.
APPENDIX E. INTERVIEW QUESTIONNAIRE (related to Chapter 6)

The following lines present the questionnaire used to carry out the interviews. The questionnaire is divided into 3 blocks:

*Common questions for all the interviewees*

1. What is your background and your experience?
2. What is your role, and what are your responsibilities within the project?
3. Which kind of documentation do you use to perform maintenance tasks: diagrams, code, textual information, etc.?
4. How do you use documentation/diagrams?
5. How often do you use the documentation?

*Block of questions for those interviewees who use UML diagrams*

6. Why do you use UML diagrams? (Give reasons) / For what purpose is UML modelling used?
7. For maintenance, do you manage (look up/ create/ modify) diagrams in a modelling tool (i.e. Enterprise Architect, Visio, etc.)? Or do you look them up in the documentation (i.e. word documents, pictures, etc.)? Did you receive any training about the tool?
8. Which diagrams do you consider to be most frequently used to perform the maintenance tasks? Which diagrams do you consider to be the most useful for performing the maintenance tasks?
9. Do diagrams help in solving defects?
   IF the answer is YES
   9.1. How do they do so?
10. When you maintain the code, do you also maintain the diagrams?
   IF the answer is YES
   10.1. How much time does it take?
   10.2. Who maintains the diagrams? (The same person who maintains the code or a different one?)
   IF the answer is NO
   10.3. Why do you not maintain the diagrams? Are the diagrams correct but not the code? Or is there another reason?
11. Do you like UML?
12. Do you think using UML has an impact on the time of the project? Do you think using UML has an impact on the quality of the final product? How?
13. What cost factors are related to using UML modelling in your work (training, tooling, etc.)?
14. Do you think there is another way in which to improve your work other than UML (i.e. another kind of diagram, etc.)?
15. Did you receive any training about UML at the Company? And before coming to the Company?
16. Do you think that the use of modelling allows errors to become incorporated?
17. Where does the diagram originate from and go to? (chain of use)
18. Do you reuse documentation from previous projects?

_Block of questions for those interviewees who do not use UML diagrams_

19. Do you use any kind of diagram to maintain the system and to communicate between team members?
20. Would you like UML diagrams to be available?
   If the answer is YES
   20.1. How do you think UML would help you to maintain the system?
   20.2. What benefits do you think UML diagrams could contribute to your work?
   20.3. Do you think UML helps to improve the quality of the final product? How?
   20.4. What cost factors are related to using UML modelling in a project?
   20.5. Do you think the size of the system influences the way in which UML is used (or not used) on a project?
   20.6. Do you think the size of the team influences the way in which UML is used (or not used) on a project?
APPENDIX F. BACKGROUND INFORMATION RELATED TO EACH INTERVIEWEE (related to Chapter 6)

Table F.1 summarizes the main background information related to each interviewee of the case study.
<table>
<thead>
<tr>
<th>Interviewee</th>
<th>ICT experience</th>
<th>Context</th>
<th>Educational field</th>
<th>Educational level</th>
<th>Gender</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Int1]</td>
<td>very high</td>
<td>common project</td>
<td>n.a.</td>
<td>school</td>
<td>male</td>
<td>project architect</td>
</tr>
<tr>
<td>[Int2]</td>
<td>medium</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>master's degree</td>
<td>male</td>
<td>project manager</td>
</tr>
<tr>
<td>[Int3]</td>
<td>very high</td>
<td>n.a.</td>
<td>electronics and mathematics</td>
<td>bachelor's degree</td>
<td>male</td>
<td>project architect</td>
</tr>
<tr>
<td>[Int4]</td>
<td>n.a.</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>bachelor's degree</td>
<td>male</td>
<td>project architect</td>
</tr>
<tr>
<td>[Int5]</td>
<td>medium</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>bachelor's degree</td>
<td>male</td>
<td>information analyst</td>
</tr>
<tr>
<td>[Int6]</td>
<td>low</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>technical lead</td>
</tr>
<tr>
<td>[Int8]</td>
<td>very high</td>
<td>n.a.</td>
<td>navy</td>
<td>n.a.</td>
<td>male</td>
<td>test engineer</td>
</tr>
<tr>
<td>[Int9]</td>
<td>high</td>
<td>outsourcing</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>delivery lead</td>
</tr>
<tr>
<td>[Int10]</td>
<td>very high</td>
<td>Embedded real-time programming</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int11]</td>
<td>very high</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>bachelor's degree</td>
<td>female</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int12]</td>
<td>low</td>
<td>migration</td>
<td>art</td>
<td>high school</td>
<td>male</td>
<td>test coordinator</td>
</tr>
<tr>
<td>[Int13]</td>
<td>very high</td>
<td>n.a.</td>
<td>n.a.</td>
<td>school</td>
<td>male</td>
<td>technical lead</td>
</tr>
<tr>
<td>[Int14]</td>
<td>high</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>n.a.</td>
<td>male</td>
<td>information analyst</td>
</tr>
<tr>
<td>[Int16]</td>
<td>n.a.</td>
<td>web/mobile projects (SCRUM)</td>
<td>electronics</td>
<td>n.a.</td>
<td>male</td>
<td>SCRUM master</td>
</tr>
<tr>
<td>[Int18]</td>
<td>very high</td>
<td>n.a.</td>
<td>chemistry and physics</td>
<td>bachelor's degree</td>
<td>male</td>
<td>system analyst</td>
</tr>
<tr>
<td>[Int19]</td>
<td>high</td>
<td>common project</td>
<td>computer sciences</td>
<td>master's degree</td>
<td>female</td>
<td>programmer / application developer</td>
</tr>
</tbody>
</table>

Table F.1. Background information of interviewees of the case study.
<table>
<thead>
<tr>
<th>Interviewee</th>
<th>ICT experience</th>
<th>Context</th>
<th>Educational field</th>
<th>Educational level</th>
<th>Gender</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Int20]</td>
<td>very high</td>
<td>n.a.</td>
<td>Business and Finances</td>
<td>bachelor's degree</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int21]</td>
<td>n.a.</td>
<td>n.a.</td>
<td>computer sciences</td>
<td>master's degree</td>
<td>male</td>
<td>analyst developer</td>
</tr>
<tr>
<td>[Int23]</td>
<td>very high</td>
<td>n.a.</td>
<td>n.a.</td>
<td>high school</td>
<td>female</td>
<td>analyst developer</td>
</tr>
<tr>
<td>[Int24]</td>
<td>very high</td>
<td>web/mobile projects (SCRUM)</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>project architect</td>
</tr>
<tr>
<td>[Int25]</td>
<td>very high</td>
<td>n.a.</td>
<td>n.a.</td>
<td>bachelor's degree</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int26]</td>
<td>very high</td>
<td>common project</td>
<td>computer sciences</td>
<td>master's degree</td>
<td>male</td>
<td>project architect</td>
</tr>
<tr>
<td>[Int27]</td>
<td>very high</td>
<td>mainframe</td>
<td>n.a.</td>
<td>HBO</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int28]</td>
<td>very high</td>
<td>old legacy system</td>
<td>psychology</td>
<td>HBO</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int29]</td>
<td>very high</td>
<td>n.a.</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>team leader</td>
</tr>
<tr>
<td>[Int31]</td>
<td>high</td>
<td>common project</td>
<td>computer sciences</td>
<td>bachelor's degree</td>
<td>male</td>
<td>deployer</td>
</tr>
<tr>
<td>[Int32]</td>
<td>very high</td>
<td>common project</td>
<td>computer sciences</td>
<td>HBO</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int33]</td>
<td>very high</td>
<td>web and mobile projects (SCRUM)</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>programmer / application developer</td>
</tr>
<tr>
<td>[Int35]</td>
<td>very high</td>
<td>change from mainframe to agile</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>information analyst</td>
</tr>
<tr>
<td>[Int36]</td>
<td>very high</td>
<td>old legacy system</td>
<td>computer sciences</td>
<td>n.a.</td>
<td>male</td>
<td>program analyst</td>
</tr>
<tr>
<td>[Int37]</td>
<td>high</td>
<td>outsourcing</td>
<td>n.a.</td>
<td>n.a.</td>
<td>male</td>
<td>project manager</td>
</tr>
</tbody>
</table>

Table F.1. Background information of interviewees of the case study.
CURRICULUM VITAE

Ana María Fernández Sáez was born in 1986 in Ciudad Real, Spain. She studied Computer Science and obtained her Bachelor Degree in Management Computer, with an itinerary of Organization and Business Administration at University of Castilla La Mancha (Spain) in 2007, and the Master degree in Computer Sciences in 2009 at the same University.

From September 2009, she worked as a Ph.D. candidate as part of the Alarcos Research Group, at the Department of Technologies and Information Systems at the University of Castilla-La Mancha (Spain) led by Dr. Mario Piattini Velthius.

Twice, she was a visiting researcher at the Software Engineering Group FaST-SE under supervision of Prof. Dr. Michel R.V. Chaudron at the University of Leiden (The Netherlands), from June 2010 until January 2011, and also during the year 2012.

During a week in May 2012, a collaboration was done with the Software Engineering Research Laboratory (SERLAB) led by Giuseppe Visaggio, in the Department of Informatics at the University of Bari (Italy).

The last collaboration was carried out with the Department of Computer Science and Engineering (CSE) of the Chalmers University of Technology at the University of Gothenborg (Sweden), during two weeks during July-August 2013 and other two weeks in December 2013.

Before, during and after her studies Ana worked for several smaller and medium Spanish IT firms including Alarcos Quality Center or Intelligent Environments. From October 2013, Ana works as a project manager at the Technical Customer Relationship Management department at Telefónica, in Madrid (Spain), focused in voice recognition projects.
SUMMARY

Nowadays, it seems that companies do not use software modelling because they fear it requires up-front investments. From an economic point of view, any investment must be justified in terms of how much payback there will be at a later stage. In the context of software projects, investment in modelling should be justified by benefits, such as improved productivity and improved product quality, which can be seen later during software development or maintenance. When such benefits are not tangible or foreseeable, modelling becomes a practice without clear added value for the system being developed.

The big issue regarding software modelling in fact resides exactly in the assumption that it will provide software projects with quantifiable benefits. The problem, therefore, is how we can investigate and prove whether or not modelling provides any benefits during software development and maintenance. As long as this question remains unanswered, it will be difficult to motivate and justify modelling activities in real software projects.

This thesis therefore contributes to partially answering these open questions by focusing the research on the benefits of using UML modelling during software maintenance.

Our approach for addressing the research questions in this dissertation is empirical in nature. The selected research methods are: a survey, a case study (including interviews) and two families of experiments. We used mixed methods research as an approach to combine quantitative and qualitative research in the same research inquiry.

Our experiments were carried out with students, whilst the survey and case study were done with professionals, to obtain realistic opinions from industrial context as regards how UML diagrams (and documentation in general) enable them to perform their maintenance tasks.

In essence, the contribution of this study is two-fold. First, this study provides sound empirical evidence about the potential benefits of UML modeling in software maintenance. As results of this evidence-based research we could highlight the following findings:

- The use of a graphical notation as a complement when attempting to understand the system that will be maintained and to support the design of the changes related to software maintenance tasks is widespread (especially UML).
- The UML adoption is directly influenced by the educational level and the experience in ICT of the maintainer, the size of the maintenance team and the size of the system under maintenance.
- UML modeling is commonly used at a low level of formality, given that the main purpose of UML models is communication and getting an overview of the system. Another reason is that models in project documentation should be understandable for all the audiences of the documentation.
Maintainers do not always use the available documentation (with or without diagrams), because use only the source code and its comments. This is owing to problems concerning a lack of synchronization caused by non-updated documentation/diagrams.

When no UML diagrams are available from the initial software development, they are rarely created during maintenance. Performing reverse engineering techniques is difficult, despite the fact that there are automatic reverse engineering tools, but a manual process is required to “clean” the diagrams. Forward designed (FD) diagrams (created during software development phase) to some extent, improve the maintenance of source code. Only in the case of junior maintainers was a better efficiency obtained when using the Reverse Engineered (RE) diagrams. This might be explained by the fact that Reversed engineered diagrams have a very high traceability with source code, so inexperienced maintainers would prefer this kind of diagrams. In the rest of the cases, Forward Designed class diagrams provide a more attractive balance between detail and relevant information than Reversed Engineered class diagrams.

The preferred Level of Detail (LoD) of diagrams depends on their purpose: there is a slight tendency in favor of using high LoD diagrams to understand the system, but there is a preference for a low LoD to maintain it. Despite the fact that the participants stated that they experience more difficulties when reading high LoD diagrams, the majority of them considered that the presence of some elements represented in the high LoD versions of each diagram is very valuable: attributes and operations in class diagrams and formal messages with parameters in sequence diagrams.

Also, several perceived benefits of using UML during software maintenance have been found during this research:
- Using UML less time is needed for a better understanding of the system being maintained; this improves the detection of defects.
- Also the reduction in the execution time of a project is considered as a benefit of UML usage.
- Moreover, when UML diagrams are available as part of the documentation, less effort is required to consult it.
- If we focus on the product quality benefits of using UML modeling, the majority of maintainers believe that it improves the quality of the ultimate software product.
- Some process benefits were also highlighted by maintainers involved in this research, such as: improved communication, the prevention of knowledge evaporation and the easing of the diagnosis of problems (especially when using behavior models). UML modeling is also considered helpful as regards improving the design before starting implementation activities (through an increased ease of peer-reviewing).

The second contribution of this Thesis is related to modeling practices. This study provides recommendations concerning best practices of modeling using UML. These recommendations are based on empirical data from an industrial case study as well as
expert opinions. Therefore, this study essentially serves as a guide and justification to continue using (and updating) the UML models of software systems, and to get the most benefits out of it.

Finally, we should stress that the benefits and costs/hurdles are difficult to quantify because they are intangible: no company keeps a record of miscommunication, poor early design choices, out-of-date or unavailable documentation and their associated costs. One of the reasons why such benefits are hard to quantify is that modeling is often one of many ways (employed in conjunction with others) of achieving a particular goal. Faults cannot, therefore, then be traced back to a single cause.

The findings obtained are a first approach to discovering the contextual factors that affect the effective use of UML during software maintenance in industrial projects. This research opens up directions for future research. For example, it would be interesting to analyze the effect of different UML notations, comparing formal diagrams with box & lines, or developing supporting tools for a proper UML documentation.

The results of this Thesis contribute to the Software Engineering body of knowledge, allowing to practitioners to have evidence to defend and promote the use of (UML) modeling in software maintenance. The recommendations provided in this Thesis can be applied in real projects of software maintenance companies, improving the daily work of their maintainers and the quality of their projects.
SAMENVATTING

Het lijkt er op dat bedrijven geen gebruik maken van software modelleren omdat ze vrezen dat er up-font investeringen voor nodig zijn. Vanuit een economisch perspectief moet voor iedere investering een rechtvaardiging bestaan die bekijkt hoeveel een investering de organisatie oplevert. In de context van software projecten, moeten investeringen in modelleren gerechtvaardigd worden door voordelen, zoals verbeterde productiviteit en verbeterde product kwaliteit, welke waargenomen kunnen worden in latere stadia van software development en onderhoud aan software. Indien zulke voordelen afwezig of onvoldoende tastbaar zijn, dan wordt modelleren een activiteit zonder duidelijk toegevoegde waarde voor het systeem dat ontwikkeld wordt.

De grote vraag rondom software modelleren ligt feitelijk in de aanname dat kwantificeerbare voordelen oplevert aan software projecten. Het probleem is daarom hoe we kunnen onderzoeken en bewijzen of modelleren wel of geen voordelen oplevert voor software onwikkeling en onderhoud. Zo lang als deze vraag onbeantwoord blijft, zal het moeilijk zijn om het gebruik van modelleren te rechtvaardigen en ertoe te motiveren.

Om deze reden draagt dit proefschrift bij aan het gedeeltelijk beantwoorden van deze open vraag door haar onderzoek te concentreren op de voordelen van het gebruik van UML modelleren bij het onderhouden van software.

Onze aanpak voor het beantwoorden van de onderzoeksvragen in dit proefschrift is empirisch van aard. De gehanteerde onderzoeksmethoden zijn: survey, case study (op basis van interviews), en twee families van experimenten. We hanteerden mixed methods research als aanpak voor het combineren van kwantitatieve en kwalitatieve bevindingen in dezelfde onderzoeksrichting.

Onze experimenten werden uitgevoerd met studenten, terwijl de enquête en case study werden uitgevoerd met professionals, om realistische meningen van beheerders uit de praktijk te verkrijgen over hoe UML-diagrammen (en documentatie in het algemeen) hen in staat stellen om hun onderhoudstaken uit te voeren.

In essentie is de bijdrage van dit proefschrift tweeledig. Ten eerste levert deze study solide empirisch bewijs over de potentiele voordelen van UML modelleren bij onderhoud van software. De bevindingen van deze study dragen hiermee bij aan de kennis op het gebied van software engineering. Op basis van ons evidence-based onderzoek presenteren we de volgende bevindingen:

- Het gebruik van een grafische notatie (in het byzonder UML) komt veelvuldig voor bij het process van begrijpen van een systeem dat onderhouden moet worden, en bij het ontwerpen van aanpassingen aan het te onderhouden systeem.
- De beslissing om UML te gebruiken hangt sterk samen met het opleidingsniveau en de ervaring van de onderhouds-ontwikkelaars, en hangt ook samen met de grootte van het onderhoudsteam en ook met de grootte van het te onderhouden systeem.
• UML wordt doorgaans gebruikt op een informele manier. Dit wordt geïnspireerd door het feit dat communicatie en het verkrijgen van een overzicht van het systeem de voornaamste doelen zijn. Een aanvullende verklaring is dat modellen begrijpelijk moeten zijn door alle gebruikers van de documentatie.

• Ontwikkelaars van software onderhoud gebruiken niet altijd de beschikbare documentatie (met of zonder diagrammen), en gebruiken enkel de source code en de daarin opgenomen toelichting. Dit is het gevolg van problemen bij het synchroon houden van de implementatie en de documentatie/modellen.

• Indien geen UML diagrammen beschikbaar zijn bij de initiële ontwikkeling van een software systeem, worden deze zelden gemaakt bij het onderhoud van dat systeem. Het toepassen van reverse engineering voor het verkrijgen van modellen is moeilijk, ondanks de beschikbaarheid van ‘reverse engineering’ tools, omdat de resulterende modellen handmatig opgeschoond moeten worden. Het gebruik van modellen die gemaakt zijn als onderdeel van het ‘forward design’ (FD) leiden in enige mate tot verbetering van het onderhoud van de source code. Enkel in het geval dat junior developers software onderhouden werd een hogere efficiency behaald met het gebruik van ‘reverse engineered’ (RE) diagrammen. Een mogelijke verklaring hiervoor is dat reverse engineered diagrammen in hoge mate corresponderen met de source code, zodat onervaren developers hier een voorkeur voor hebben. In de andere gevallen bieden forward designed class diagrammen een aantrekkelijkere balans tussen detail en relevante informatie dan reverse engineered class diagrammen.

• Het te prefererent niveau van detail (LoD) voor het beschrijven van diagrammen hangt af van de beoogde doelen van hun gebruik: er bestaat een lichte voorkeur voor het gebruik van hoog niveau van detail voor het begrijpen van een systeem, maar er is een voorkeur voor een laag niveau van detail voor het onderhouden van een systeem. Ondanks het feit dat ontwikkelaars stelden dat ze meer moeite hadden met het lezen van diagrammen met hoog niveau van detail, beschouwden de meerderheid van hen het waardevol dat enkele elementen van diagrammen in hoog niveau van detail worden weergegeven: attributen en operaties in class-diagrams en formele messages met parameters in sequence diagrammen.

• Enkele vermeende voordelen van het gebruik van UML tijdens onderhoud aan software zijn naar boven gekomen:
  o Bij het gebruik van UML is minder tijd benodigd voor het verkrijgen van een beter begrip van het te onderhouden systeem; het gebruik van UML leidt ook tot verbeterde opsporing van fouten in de software.
  o De reductie in de doorlooptijd van projecten wordt beschouwd als een voordeel van
  o het gebruik van UMLWanneer we kijken naar de voordelen bij de kwaliteit van het software product, dan meent de meerderheid van de software onderhoud-ontwikkelaars dat het gebruik van UML modelleren de kwaliteit van het uiteindelijke product verbeterd.
  o Wanneer we kijken naar de voordelen bij het process van het ontwikkelen van de software, dan melden software-onderhouds developers de volgende
voordelen van het gebruik van UML modellen: verbeterde communicatie, het voorkomen van de verdamping van kennis, het vereenvoudigen van de diagnose van problemen (in het bijzonder rondom gedrags-modellen). UML modelleren wordt ook behulpzaam gevonden bij het verbeteren van het ontwerp van het systeem voordat aan de implementatie begonnen wordt (door het vergemakkelijken van peer-reviewing).

De tweede bijdrage van dit proefschrift is gerelateerd aan de beoefening van modelleren. Dit onderzoek geeft aanbevelingen voor ‘best practices’ bij UML modelleren. Deze aanbevelingen zijn gebaseerd op empirisch onderzoek bij zowel een industriële case studie, als ook op de meningen van experts. Dit onderzoek dient als gids en rechtvaardiging voor het voortzetten van het gebruik en het bijwerken van UML modellen, en voor het behalen van de voordelen van UML modelleren.

Ter afsluiting benadrukken we dat de voor- en nadelen van het gebruik van UML moeilijk te kwantificeren zijn omdat ze niet tastbaar zijn: geen enkel bedrijf houdt bij welke problemen ontstonden door miscommunicatie, door slechte ontwerp-keuzes, door achterstallige- of onbeschikbare documentatie. Een van de redenen dat dergelijke aspecten moeilijk te kwantificeren zijn, is dat modelleren van een van meerdere manieren is voor het bereiken van bepaalde doelstellingen. Om deze reden kunnen fouten niet teruggevoerd worden op een enkele oorzaak.

De verkregen bevindingen zijn een eerste aanzet voor het ontdekken van de contextuele factoren die van invloed zijn op het gebruik van UML bij het onderhoud van software in industriële projecten. Dit onderzoek heeft enkele vragen voor vervolgonderzoek opgeleverd: een voorbeeld hiervan is de vraag naar het effect van verschillende UML notaties, waarbij formele en ‘box-and-lines’ vergeleken worden. Ook geeft dit onderzoek richtingen aan voor het ontwikkelen van tools die het gebruik van UML gedurende het software ontwikkeltraject beter ondersteunen.

De resultaten van dit onderzoek dragen bij aan de kennis over Software Engineering, en stellen software ontwikkelaars in de praktijk in staat om op basis van empirische bevindingen het gebruik van UML modellering te bevorderen en verdedigen. De aanbevelingen van dit proefschrift kunnen worden toegepast in de industriële praktijk van software onderhoud, en hiermee leiden tot verbetering van hun dagelijkse werk en verbetering van de kwaliteit van hun projecten.
ACKNOWLEDGMENTS

First of all, I would like to express my gratitude to my supervisors Marcela Genero Michel Chaudron and Joost Kok for their technical and non-technical advice during the realization of the Thesis, which would not exist without their help and confidence. Also for sharing their personal life with me. I will always remember our great moments at conferences.

My thanks go also to the universities that have made this research possible: The University of Castilla-La Macha and the University of Leiden. Furthermore, I would like to thank the universities that have allowed me to develop different research studies in their installations (thanks especially to Danilo Caivano at the University of Bari, Italy and Isabel Ramos at the University of Seville, Spain). I am also very grateful to the research groups at those universities that have given me the opportunity to carry out different research stays and to those people who helped me in any way: reviewing papers, participating in the trial and the real studies, sharing the room at the university, or just sharing a coffee.

I wish to extend my thanks to the Alarcos Research Group and LIACS, and all their kind members, for providing me with all the human and technical resources that were necessary to finish this Thesis. My greatest thanks go to all my PhD colleagues from both institutions.

I would like to thank all the software architects, developers, designers, project leaders, testers and other people that I interviewed and surveyed, and all the students that participated in my experiments. Thank you for your time and openness.

Personally, many friends have been very supportive. I thank all of them for being in the right place and at the right time, especially Belen. Also thanks to Lucia for designing this beautiful cover.

Of course, I would like to thank my family: Bea, Vicky, Alba, Maria Angeles and my father. Thanks for your unconditional support. And a very special thanks to my mother. She was the person who encouraged me to make to this journey, who was always there supporting me, and who (unfortunately) had to see the final result of this adventure from heaven. I hope that although it has taken more than enough time, she is proud of me.

Finally, I would like to thank Jose Carlos. Thanks for making my life easier and for helping me with the hard work that this thesis has involved. Thanks for making sure that I started and completed this project. Lots of thanks for your inexhaustible patience, and for making me happy every day.

En primer lugar, me gustaría expresar mi gratitud a mis supervisores Marcela Genero, Michel Chaudron and Joost Kok por su asesoramiento técnico y no técnico durante la realización de esta Tesis, que no existiría sin su ayuda y confianza. También
para compartir su vida personal conmigo. Siempre recordaré nuestros grandes momentos en los congresos.

Mi agradecimiento también a las universidades que han hecho posible esta investigación: la Universidad de Castilla-La Macha y la Universidad de Leiden. Además, me gustaría agradecer a las universidades que me han permitido desarrollar diferentes estudios de investigación en sus instalaciones (gracias especialmente a Danilo Caivano en la Universidad de Bari, Italia, e Isabel Ramos en la Universidad de Sevilla, España). También estoy muy agradecida con los grupos de investigación de aquellas universidades que me han brindado la oportunidad de llevar a cabo diferentes estancias de investigación y con aquellas personas que me ayudaron de alguna manera: revisando documentos, participando en ensayos y en estudios reales, compartiendo su despacho en la universidad, o simplemente compartiendo un café.

Deseo extender mi agradecimiento al Grupo de Investigación Alarcos y LIACS, y a todos sus amables miembros, por proporcionarme todos los recursos humanos y técnicos que fueron necesarios para finalizar esta Tesis. Mi mayor agradecimiento a todos mis compañeros de doctorado de ambas instituciones.

Me gustaría agradecer a todos los arquitectos de software, desarrolladores, diseñadores, líderes de proyectos, testers y otras personas que entrevisté y encuesté, y a todos los estudiantes que participaron en mis experimentos. Gracias por su tiempo y su mente abierta.

Personalmente, muchos amigos me han apoyado mucho. Muchas gracias a todos por estar en el lugar correcto y en el momento correcto, especialmente a Belén. También a Lucía por diseñar esta preciosa portada.

Por supuesto, me gustaría dar las gracias a mi familia: Bea, Vicky, Alba, María Angeles y mi padre. Gracias por su apoyo incondicional. Y un agradecimiento muy especial a mi madre. Ella fue la persona que me animó a embarcarme en este viaje, que siempre estuvo ahí apoyándome, y que (desafortunadamente) ha tenido que ver el resultado final de esta aventura desde cielo. Espero que, aunque haya tomado más tiempo del previsto, esté orgullosa de mí.

Finalmente, me gustaría darle mi agradecimiento a Jose Carlos. Gracias por facilitarme la vida y por ayudarme con el arduo trabajo que ha supuesto esta tesis. Gracias por asegurarte de que comencé y terminé este proyecto. Muchas gracias por tu inagotable paciencia y por hacerme feliz todos los días.